CAPITOLUL 1

Componentele de bază ale sistemelor de calcul

CUPRINS

CAPITOLUL 1 ---------- **1**

1.1. SISTEMUL DE CALCUL ---------- **3**
1.1.1. Definiţii ---------- *3*
1.1.2. Aplicaţiile sistemelor de calcul ---------- *3*
1.1.3. Sistemul de calcul – din diferite puncte de vedere ---------- *4*
1.1.4. Structura ierarhică de organizare a calculatorului ---------- *6*
1.1.5. Structura funcţională a unui sistem de calcul ---------- *7*
1.1.6. Performanţele unui sistem de calcul ---------- *9*
1.1.7. Clasificări ---------- *9*
1.1.8. Principiile von Neumann ---------- *12*
1.1.9. Modelul formal al unui calculator ---------- *13*
1.1.10. Evoluţia sistemelor de calcul ---------- *14*

1.2. SISTEMUL DE INTRARE/IEŞIRE. DISPOZITIVE PERIFERICE ---------- **18**
1.2.1. Clasificări ---------- *19*
1.2.2. Caracterizarea unor periferice ---------- *20*
1.2.2.1. Discurile dure. Hard-discul ---------- 20
1.2.2.2. Discurile flexibile. Discheta ---------- 21
1.2.2.3. Compact discurile ---------- 22
1.2.2.4. Monitoarele (*graphic display*) ---------- 22
1.2.2.5. Principalele adaptoare video ---------- 24
1.2.2.6. Mouse-ul ---------- 25
1.2.2.7. Reţele de comunicare ---------- 26
1.2.2.8. Placa de reţea ---------- 27
1.2.2.9. Modemul ---------- 28
1.2.3. Canalul de intrare / ieşire ---------- *28*
1.2.4. Legarea perifericelor la SC. Unităţile de interfaţă ---------- *28*
1.2.5. Magistrale ---------- *30*
1.2.6. Transferul eficient de date. Procesoare specializate ---------- *35*

1.3. UNITATEA DE MEMORIE ---------- **38**
1.3.1. Structura fizică a memoriei ---------- *38*
1.3.2. Structura ierarhică de organizare a memoriei ---------- *39*
1.3.3. Memorii cache ---------- *42*
1.3.4. Memoria virtuală ---------- *43*
1.3.5. Memorii cu acces aleator – RAM şi ROM ---------- *45*

1.4. PROIECTAREA CALCULATOARELOR MODERNE ---------- **47**

1.1. Sistemul de calcul

1.1.1. Definiții

Sistemul de calcul (SC, *System Computer* sau calculator) este reprezentat de o structură destinată prelucrării datelor.

Adesea, un SC este descris prin cele două subsisteme componente: **hardware** şi sistemul **software**.

Partea **hardware** constă din toate componentele electronice care alcătuiesc partea fizică a SC. Sistemul **software** constă din instrucțiuni şi date pe care SC le prelucrează pentru a executa diversele cerințe ale utilizatorului. O secvență de instrucțiuni ale SC se numeşte **program**. Datele prelucrate de SC constituie **baza de date**.

Totalitatea componentelor sistemului de calcul solicitate de programe pentru executarea lor poartă numele de **resurse**.

Concret, **SC** este format din: resurse fizice, resurse logice şi resurse informaționale. Avem următoarele corespondențe:

– **hardware = resurse fizice,** adică procesoare, memorii, dispozitive de intrare/ieşire,

– **software = resurse logice.** Acestea se împart în software pentru sistem (*systems software*) şi software pentru aplicații (*applications software*). **Soft-ul pentru sistem** constă în aplicațiile sistem care oferă servicii pentru uz comun. Din această categorie fac parte: sistemul de operare, compilatoarele, asambloarele, s.a. **Soft-ul pentru aplicații** constă în aplicațiile propriu-zise, adică: programe utilitare, medii de programare, programele utilizatorilor. Aplicațiile sistem sunt la granița dintre hardware şi aplicațiile propriu-zise.

Observație. Trebuie precizat că această clasificare este una relativă deoarece un anumit program nu este obligatoriu să facă parte exclusiv dintr-o astfel de categorie. De exemplu, un compilator este o aplicație sistem, dar, mai mult, poate face parte dintr-un sistem de operare, dacă este vândut împreună cu acesta şi depinde de acel sistem de operare.

Un sistem de operare este un set de programe care gestionează toate resursele sistemului de calcul, astfel încât programele care funcționează pe acel sistem să se execute la parametrii maximi (utilizare completă şi optimă).

– **resurse informaționale,** adică suporturi de memorie externă pentru organizarea şi stocarea datelor.

1.1.2. Aplicațiile sistemelor de calcul

Pe bună dreptate se spune despre calculator că este *maşina care a revoluționat lumea*. Această afirmație o susținem în continuare arătând diversitatea domeniilor în care calculatorul / procesorul este nelipsit.

Primul exemplu ar putea fi maşina automată destinată tranzacțiilor bancare (**bancomatul**). De neconceput în anii '50 — când cel mai ieftin

calculator costa 500.000 $ — astăzi, bancomatele te întâmpină la tot pasul, oferind servicii sigure, toate operațiile fiind coordonate de un procesor central.

In **industria automobilelor**, de când microprocesorul a scăzut spectaculos în volum şi i-au crescut realmente performanțele (începutul anilor '80), procesorul este nelipsit în sistemele destinate scăderii poluării, a creşterii randamentului conbustibililor şi în controlul reacției pungilor de aer în cazul coliziunilor vehiculelor (*air-bag*).

Creşterea performanțelor calculatoarelor simultan cu micşorarea dimensiunilor componentelor interne a condus la posibilitatea dezvoltării unei industrii paralele, cea a **calculatoarelor portabile** (*laptop*-uri).

In sfârşit, dar nu în ultimul rând, este de remarcat larga răspândire a calculatoarelor pe masa de lucru a tot mai multor persoane. La acest aspect a contribuit, credem în mare măsură, şi tentația de a avea acces la populara rețea de comunicare şi informare care este **Internet**-ul. In general, rețeaua Internet foloseşte aplicații WWW (*World Wide Web*), adică un set de programe care formează un sistem de comunicare bazat pe informații *hypertext*. WWW funcționează prin comunicare de date pe baza unui model client/server. Prin *hypertext* înțelegem un sistem de grupare asociativă a informațiilor rezidente pe mai multe servere Web, astfel încât să ofere utilizatorului senzația că el consultă un singur document. Limbajul de descriere a *hypertext*-ului este HTML (HyperText Markup Language). Programele care permit vizualizarea (consultarea) paginilor Web sunt *browser*-ele. Un *browser* este deci un client Web care are acces la informații multimedia şi multiprotocol oferite de serverul Web accesat.

Dintre aplicațiile frecvente ale WWW amintim: distribuirea ştirilor, înscrierea într-un grup cu interese comune, consultarea unor note bibliografice sau referințe ştiințifice, comandarea şi trimiterea cadourilor prin cataloage on-line, orientarea electronică pentru itinerariul de vacanță, s.a.m.d. In plus, există marele avantaj al accesului de pe calculatorul de acasă pe orice alt calculator conectat la Internet. Astăzi, nimeni nu mai cară calculatoare…

1.1.3. Sistemul de calcul – din diferite puncte de vedere

Descrierea operațiilor unui SC la nivel de *hardware* diferă în funcție de scopul urmărit. Astfel, un SC poate fi privit cel puțin din trei puncte de vedere:

1. organizarea componentelor;
2. construirea;
3. arhitectura internă.

Organizarea SC se referă la cunoaşterea modului în care operează fiecare dintre componentele hard şi posibilitățile lor de interconectare astfel încât să formeze SC performant.

Concret, din acest punct de vedere, trebuiesc descrise:

a) circuitele logice combinaționale, secvențiale şi/sau integrate care reprezintă structura internă a fiecărei componente hard;

b) modurile de reprezentare a datelor în memoria SC;

c) operațiile fundamentale într-un SC.

Construirea SC (design) se referă la acțiunea de determinare a tipului componentelor hard care trebuiesc folosite şi a tipurilor de interconectare a acestora, astfel încât SC construit să răspundă cerințelor utilizatorului.

Concret, din acest punct de vedere, trebuie să ne referim la:

a) modurile de reprezentare a instrucțiunilor;

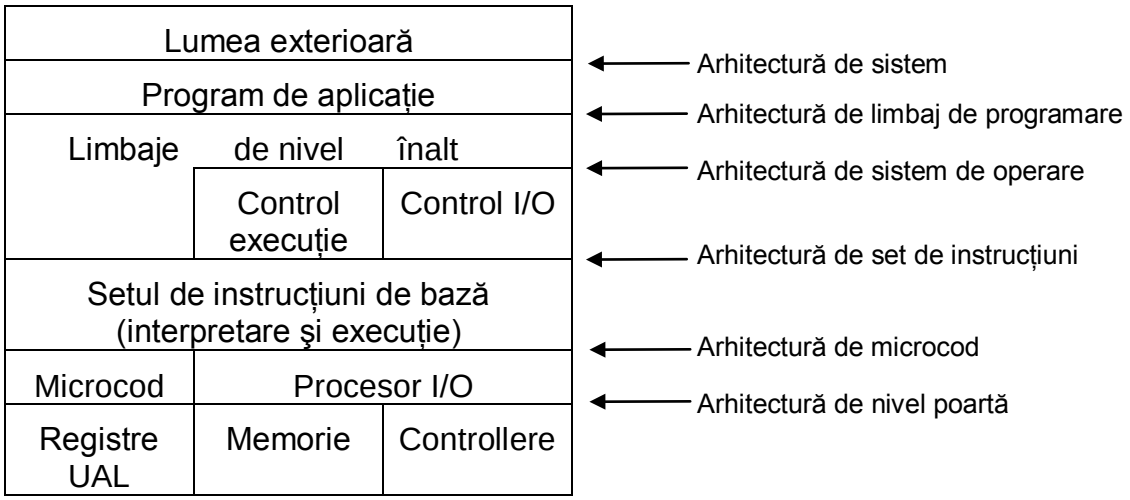
b) controlul instrucțiunilor prin programe;

c) fundamentele limbajului maşină şi ale limbajului de asamblare.

Arhitectura sistemului de calcul se referă la structura şi comportarea SC văzut de utilizator. Când ne referim la arhitectura unui SC trebuie să dăm specificațiile diferitelor unități funcționale şi conectările fizice şi logice dintre ele.

Conform DEX (Dicționarului Explicativ al limbii române), **arhitectura calculatoarelor** se referă la asamblarea subsistemelor funcționale într-un calculator numeric, pentru a obține performanțele impuse în condiții de cost avantajoase.

Conform [13], sintagma *arhitectura calculatoarelor* este adesea utilizată cu semnificația simplă de *organizarea şi proiectarea calculatoarelor*. Detaliind din punct de vedere practic, într-un sistem de calcul se pot distinge mai multe nivele de arhitectură (vezi figura următoare), fiecare definit de legătura pe care o stabileşte între subsistemele sistemului gazdă.



In general, prin *arhitectură* ne referim la interfața dintre două astfel de module funcționale. **Arhitectura de sistem** sau **arhitectura sistemului de calcul** este cea dintre sistemul de calcul şi mediul exterior lui. La limita dintre *hardware* şi *software* se află arhitectura setului de instrucțiuni care reprezintă nivelul elementar la care sunt decodificate şi executate instrucțiunile recunoscute de unitatea de prelucrare.

1.1.4. Structura ierarhică de organizare a calculatorului

Din punctul de vedere al rolului fiecărei resurse a sistemului de calcul putem privi organizarea internă a calculatoarelor moderne pe două sau mai multe nivele. Fiecare nivel este construit pe baza predecesorului său şi reprezintă o abstractizare distinctă, cu obiecte şi operații specifice.

In general, se pot identifica următoarele şase nivele conceptuale:

Nivelul 0. Nivelul logic digital. La acest nivel interesează **porțile logice** care intră în structura fiecărei componente interne a calculatorului. Fiecare poartă este construită dintr-un număr mic de tranzistoare. Câteva porți pot fi combinate pentru a forma un bistabil (o memorie de 1 bit), care poate stoca 0 sau 1. Memoriile de 1 bit pot fi combinate pentru a forma regiştri. In acest curs, nivelul logic digital este subiectul capitolului al doilea.

Nivelul 1. Nivelul microarhitecturii. La acest nivel calculatorul este o colecție de **regiştri** (care formează memoria locală) şi un circuit capabil să execute operații aritmetice şi logice (numit UAL, **unitate aritmetico-logică**). Regiştri sunt conectați la UAL pentru a forma o cale de date (engl. *datapath*) prin care se transferă datele între regiştrii şi UAL. Operația de bază la nivelul căii datelor este selecția regiştrilor asupra cărora operează UAL. După felul în care este controlat transferul datelor deosebim maşini de calcul cu control microprogramat sau maşini de calcul cu control bazat şi pe circuite ale componentelor hardware.

Nivelul 2. Nivelul arhitecturii setului de instrucțiuni sau nivelul ISA (engl. *Instruction Set Architecture*)**.** Fiecare producător de calculatoare publică un manual pentru maşinile pe care le vinde, manual care se referă la acest nivel ISA, dar nu şi la nivelele de dedesubt. In acest manual producătorul descrie **limbajul maşinii** pe care o vinde, adică setul de instrucțiuni maşină recunoscute de maşina respectivă şi pe care, implicit, aceasta le poate executa. Această descriere constă în detalierea instrucțiunilor executate în mod interpretat de către componenta de control (microprogram sau circuite hardware).

Nivelul 3. Nivelul maşină al sistemului de operare. La acest nivel se detaliază structura şi funcțiile sistemului de operare al maşinii. Față de nivelul ISA, sunt adăugate instrucțiuni noi, există o organizare diferită a memoriei, se remarcă posibilitatea execuției paralele şi/sau concurente a programelor (task-urilor).

Observație. In general, nivelele de la 1 la 3 nu sunt proiectate pentru a fi utilizate de programatorul obişnuit. Ele sunt un suport pentru aplicațiile dezvoltate la nivele superioare de către programatorii de aplicații. O altă diferență importantă între nivelele 1-3 pe de o parte şi nivelele 4,5 pe de altă parte este natura limbajului oferit. Limbajele maşină ale nivelelor inferioare sunt numerice (programele sunt secvențe greoaie de numere binare), în timp ce programele dezvoltate la nivelele superioare folosesc limbaje care conțin cuvinte uzuale sau abrevieri uşor de înțeles.

Nivelul 4. Nivelul limbajului de asamblare. Acest nivel oferă programatorilor posibilitatea de a scrie programe pentru nivelele inferioare

într-o formă simbolică, mai accesibilă decât formatul impus de limbajul maşină. Acest nivel face obiectul capitolului al patrulea din curs.

Nivelul 5. Nivelul limbajului orientat pe problemă. La acest nivel sunt proiectate limbaje pe care le vor folosi programatorii de aplicații care au de rezolvat probleme specifice. Astfel de limbaje sunt uzual numite **limbaje de nivel înalt**.

Mulțimea de tipuri de date, operații şi caracteristici ale fiecărui nivel se numeşte **arhitectura** nivelului respectiv. Cu alte cuvinte, arhitectura se referă la acele aspecte care sunt vizibile pentru utilizatorul unui nivel.

Conform Tanenbaum [7] **arhitectura calculatoarelor** se referă la studiul proiectării acelor părți ale unui sistem de calcul care sunt vizibile programatorului. In general, arhitectura calculatoarelor şi organizarea calculatoarelor sunt sintagme sinonime.

1.1.5. Structura funcțională a unui sistem de calcul

Intr-un SC, resursele fizice împreună cu cele logice cooperează pentru satisfacerea cerințelor utilizatorilor în ceea ce priveşte: *introducerea* (recepționarea) datelor, *memorarea* (conservarea) datelor şi informațiilor, *prelucrarea* informațiilor, *transmiterea* informațiilor la alte sisteme de calcul şi *regăsirea* informațiilor.

Indeplinirea acestor operații cade în sarcina unor subsisteme, numite **unități funcționale** ale SC. Acestea sunt conectate fizic şi logic între ele şi se individualizează prin funcția specifică fiecăreia în sistemul de calcul.

După funcțiile pe care le îndeplinesc, unitățile funcționale se grupează în următoarele clase, formând *structura fizică* a sistemului de calcul:

– **unități de schimb** — pentru recepționare şi transmitere de informație. Unitățile de schimb formează **componenta de control şi comandă (UCC).**

– **componentele unității aritmetico-logice (UAL)** — pentru executarea operațiilor aritmetice şi logice. Acestea preiau din memorie valorile operanzilor şi depun tot în memorie rezultatele.

– **procesoare** — pentru prelucrarea datelor. Procesoarele din SC formează nucleul pentru **unitatea centrală de prelucrare** (**UCP** sau CPU = Central Processing Unit).

– **memorii interne** — pentru stocarea datelor şi instrucțiunilor;

– **dispozitive periferice** — pentru preluarea şi/sau transmiterea informațiilor externe. Perifericele formează **sistemul de intrare/ieşire**.

Cu aceste componente se pot scrie ecuațiile care reprezintă *structura logică* a sistemului de calcul. Acestea sunt:

UCC+ UAL= UCtrlP

UCtrlP+Regiştrii = UCP

UCP+ Memorii = UC

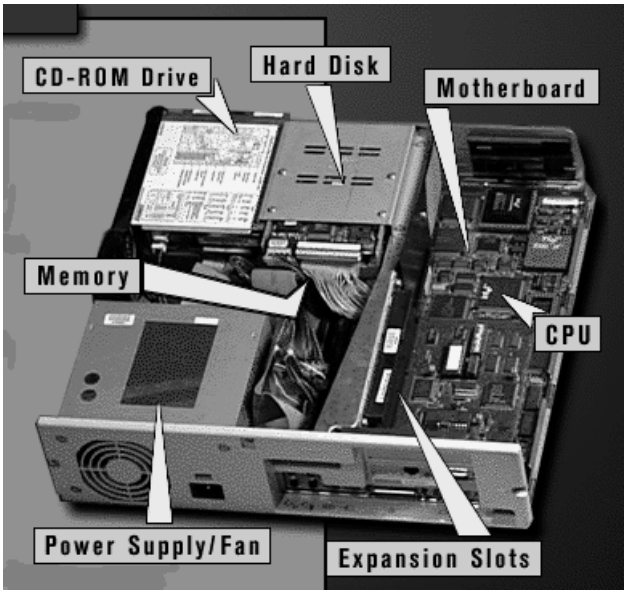
UC+ IOS = SC

unde:

- ✓ **UCtrlP** este **unitatea de control şi prelucrare**
- ✓ **UC** este **unitatea centrală**
- ✓ **IOS** este **sistemul de intrare/ieşire**.

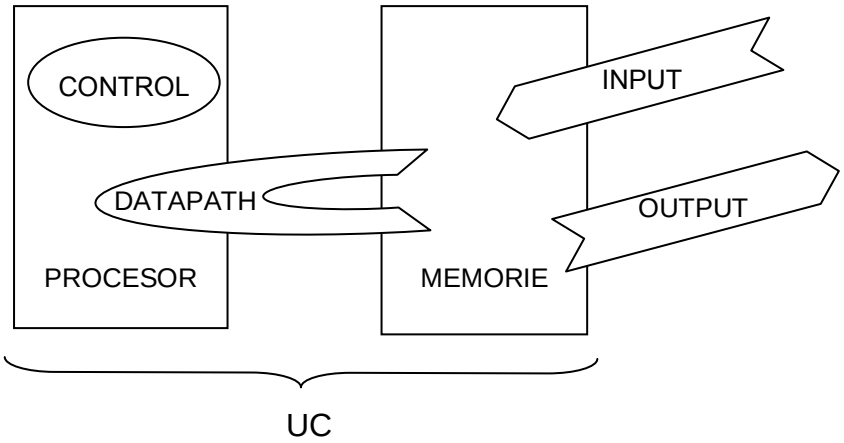
UC este unitatea de control a întregului sistem de calcul: supervizează activitatea oricărei componente şi decide în situații conflictuale.

Aceste componente se regăsesc în figura următoare:



Orice componentă a sistemului de calcul poate fi încadrată în una dintre categoriile: MEMORII, INPUT, OUTPUT, CONTROL, DATAPATH.

Relațiile de dependență între unitățile funcționale ale SC vor fi detaliate ulterior, în paragraful referitor la *etapele prelucrării instrucțiunilor unui program.* Prezentăm aici numai schema orientativă a acestor dependențe:



De aici rezultă că procesorul preia instrucțiunile şi datele din memorie, în timp ce controlul trimite semnale care determină operații pe Datapath, în

memorii şi pe dispozitivele periferice. Datapath (calea datelor) include registre, UAL şi mai multe magistrale de legătură (vezi şi exemplul de organizare a registrelor). Cu alte cuvinte, *datapath* este acea parte a UCP care conține UAL şi intrările şi ieşirile acesteia.

1.1.6. Performanțele unui sistem de calcul

Performanțele unui sistem de calcul se apreciază în funcție de:

- **viteza de execuție,** dată de numărul instrucțiunilor executate în unitatea de timp;
- **capacitatea memoriei**, dată de numărul de octeți care pot fi memorați în sistem, atât în memoria internă cât şi în memoria externă;
- **timpul de acces la memorie** egal cu timpul necesar obținerii unei unități de informație memorată (octet, cuvânt)
- **fiabilitatea**, dată de numărul de defecte (hardware sau software) apărute într-un interval de timp;
- **tipul arhitecturii** reprezentat de modalitatea de funcționare a sistemelor (multiprogramare, timp partajat, timp real, acces la distanță, multiprocesare).

1.1.7. Clasificări

Calculatoarele se pot clasifica după felul în care se memorează datele, după structura lor arhitecturală sau, o a treia variantă, după performanță (mărime, posibilități de prelucrare, preț şi viteză de operare). Vom descrie în continuare grupele de calculatoare din aceste categorii.

După felul în care se memorează datele în sistemele de calcul avem:

– **calculatoare numerice** (la care ne vom referi în continuare). Datele şi informațiile sunt codificate sub formă numerică binară;

– **calculatoare analogice**: sistemul de calcul este reprezentat printr-un sistem fizic în care pot fi stabilite relații matematice dinainte prescrise între variabilele continue ale sistemului fizic; spre deosebire de cazul digital, maşinile analogice nu pot fi caracterizate numai prin două stări. In general, o maşină analogică transformă o problemă matematică (teoretică sau practică) într-una bazată pe mărimi fizice (segmente, unighiuri, intensitatea curentului electric, variații de potențial) pe baza unei analogii. In final se obține un rezultat aproximativ, dar convenabil din punct de vedere practic.

Observație. În această clasificare nu ne-am referit la sistemele analogice, caracterizate de semnale sau mărimi analogice (semnale sau mărimi care variază continuu), cum este transmiterea semnalelor de telefonie.

– **calculatoare hibride** rezultate din cuplarea unui calculator numeric cu unul analogic.

– **Din punct de vedere arhitectural, avem:**

– sistem de calcul monoprocesor;

- sistem de calcul multicalculator;
- sistem de calcul multiprocesor.

Un sistem de calcul monoprocesor are o singură unitate centrală de prelucrare (CPU).

Un sistem de calcul multicalculator include cel puțin două unități centrale, fiecare dintre calculatoarele membre având un grad ridicat de independență.

La multicalculatoare divizarea sarcinilor de prelucrare se face cu interacțiune doar la nivelul datelor supuse prelucrării.

Modelul uzual al sistemelor multicalculator sunt rețelele de calculatoare.

Un sistem de calcul multiprocesor conține cel puțin două unități de prelucrare cu acces comun la memoria internă unică.

Procesele membre ale sistemului interacționează atât la nivel hard cât şi la nivel soft, dar soft-ul de bază (sistemul de operare) este unic.

Prin varianta arhitecturală multiprocesor se asigură o mărire a capacității de prelucrare prin posibilitatea de executare în paralel a unor sarcini distincte.

Totalitatea resurselor unui sistem multicalculator sau a unuia multiprocesor formează un **sistem distribuit**.

Analizate după mărime, posibilități de prelucrare, preț şi viteză de operare distingem:

- supercalculatoare;
- sisteme de calcul medii-mari (*main-frame*);
- minicalculatoare;
- microcalculatoare.

Supercalculatoarele conțin mai mult de 8 UCP care pot să execute mai mult de un bilion de instrucțiuni pe secundă. De aceea, ele sunt folosite în prelucrări extrem de complexe. De obicei sunt sisteme multiprocesor.

Sistemele de calcul medii-mari (main frame) conțin mai multe UCP alături de unități principale de intrare/ieşire care operează cu viteză înaltă. Astfel, aceste calculatoare sunt apreciate în aplicațiile care solicită viteză de operare foarte ridicată.

Pentru folosirea acestor calculatoare sunt necesare instalații şi proceduri de menținere în funcționare permanentă.

Exemplu. În această categorie se încadrează calculatorele româneşti din familia FELIX.

Minicalculatoarele execută operații specializate. Ele sunt folosite în aplicații multiutilizator, în prelucrări de texte sau ca şi componente ale maşinilor de control numeric.

Datorită facilităților oferite în cadrul proceselor de comunicații, minicalculatoarele sunt frecvent folosite în transmisii de date între sisteme dispersate geografic.

Actualmente folosirea lor este în scădere datorită utilizării microcalculatoarelor.

De obicei sunt sisteme interactive multiutilizator.

Exemple. În România s-au produs minicalculatoarele CORAL şi INDEPENDENT. Sistemele VAX produse în Statele Unite sunt încă actuale prin noile lor variante.

Microcalculatoarele (PC sau **Personal Computer)** au apărut datorită dezvoltării tehnicii de calcul şi a informaticii.

La aceste sisteme de calcul unitatea centrală este implementată cu un microprocesor.

Microcalculatoarele sunt sisteme interactive, monoutilizator.

Calculatoarele personale cunosc o dezvoltare rapidă şi o diversificare continuă datorită avantajelor față de celelalte categorii, dintre care amintim:

- ✓ cunoştințele de informatică necesare unui utilizator de PC sunt minimale;
- ✓ folosirea unui astfel de calculator pentru activități zilnice se poate realiza chiar fără o pregătire de specialitate; aceasta şi datorită interfețelor grafice ale sistemelor de operare şi aplicațiilor care realizează legătura între utilizator şi sistemul de calcul;
- ✓ au gabarit redus, pot fi chiar portabile;
- ✓ au aplicabilitate în orice domeniu de activitate.

Exemple. Microcalculatoare produse în România: aMIC, HC-85, TIM-S, CIP, HC-90, FELIX PC. În Marea Britanie s-au remarcat microcalculatoarele de tip Sinclair Spectrum, iar în Statele Unite, Commodore 64, Apple II şi toate variantele de sisteme compatibile IBM PC.

Prezentăm în continuare clasificarea dată de Flynn în 1972 pentru SC. Aceasta priveşte calculatoarele **din punctul de vedere al mărimii fluxului instrucțiunilor şi mărimii fluxului datelor**. Astfel, fluxurile pot fi:

SISD (**S**ingle **I**nstruction **S**ingle **D**ata Stream) pentru toate sistemele de calcul cu o unitate centrală;

SIMD (**S**ingle **I**nstruction **M**ultiple **D**ata Stream) pentru unele supercalculatore şi toate SC care au o singură unitate centrală cu n unități aritmetico-logice şi n module de memorie ataşate unității centrale interconectate;

MISD (**M**ultiple **I**nstruction **S**ingle **D**ata Stream): categorie vidă, deoarece combinația aceasta de fluxuri este ineficientă;

MIMD (**M**ultiple **I**nstruction **M**ultiple **D**ata Stream) pentru sistemele multiprocesor sau multicalculator. De exemplu, multe supercalculatoare şi sistemele distribuite.

Observație. Deşi Flynn a apreciat ineficientă varianta MISD, astăzi există modele paralele care se încadrează în această categorie. Este vorba despre execuția în *pipeline* (linie de asamblare).

În încheierea acestui paragraf dăm un tabel sintetic al **generațiilor de calculatoare**, aşa cum au apărut ele cronologic (M = capacitate de memorie, V = viteză de prelucrare în instrucțiuni pe secundă).

Gen	Perioada	CPU	Memorie	Limbaje	Memorie. CPU
I	1946 – 1956	Tuburi electronice	Tambur magnetic	Limbaj de asamblare	M: 2KO; V: 1E4 i/s
II	1957 – 1963	Tranzistori	Inele de ferită	Limbaje de nivel înalt: FORTRAN, COBOL	M: 32 KO; V: 2E5 i/s
III	1964 – 1981	Circuite integrate	Memorii semicond.; Discuri magnetice	Limbaje de nivel foarte înalt (PASCAL, LISP, limbaje grafice)	M: 2 MO; V: 5E6 i/s
IV	1982 – 1989	Circuite integrate pe scară largă şi foarte largă	Memorii cu bule; Discuri optice	ADA, limbaje orientate obiect	M: 8MO; V: 3E7 i/s pentru supercalculatoare
V	după1990	Circuite integrate pe scară extrem de largă; Masini LISP şi PROLOG	Arhitecturi paralele	Limbaje concurente; Limbajul natural; Limbaje funcționale (LISP); Limbaje logice (PROLOG)	V: de la 1E9 i/s la 1E12 i/s; M: suficientă pentru stocarea şi prelucrarea cunostintelor (inteligența artificială); Tehnologia vorbirii

1.1.8. Principiile von Neumann

Începând cu 1940, după apariția lucrărilor teoretice privind calculele automate, lucrări datorate în principal lui Alan Turing, lumea cercetătorilor a devenit tot mai preocupată de construcția efectivă a unor calculatoare. Matematicianul american John von Neumann a intuit, prin analogie cu anatomia creierului uman, principiile care ar trebui să stea la baza construcției calculatoarelor.

Enumerăm aceste principii, aşa cum au fost formulate la vremea respectivă.

1. Deoarece maşina este destinată calculelor, ea execută cele patru operații aritmetice. Se impune deci, existența unei componente specializate pentru calcule. (Astăzi, aceasta este numită *unitatea aritmetico-logică*.)

2. Operațiile se vor executa secvențial. Pentru a asigura elasticitatea şi o relativă universalitate, se impune o distincție între instrucțiunile necesare rezolvării unei probleme particulare şi controlul

general asupra unor instrucțiuni executate la un moment dat. (Apare, astfel, ceea ce astăzi numim *componentă de control*.)

3. Este necesară o componentă numită *memorie internă* în care „se țin minte" pentru un timp limitat atât instrucțiunile, cât şi datele necesare rezolvării problemei.

Cele trei componente, luate împreună, sunt analoage *neuronilor* din sistemul nervos central.

Se impune, continuând analogia, să existe un senzor care să recepționeze semnalele de orice fel provenite din exterior. Este necesar, de asemenea, un motor care să acționeze asupra mediului exterior. (Astăzi, acestea se numesc *dispozitive de intrare/ieşire.*)

4. Este necesară o componentă care să „țină minte permanent", să înregistreze aşa cum omul îşi notează ceea ce nu doreşte să uite. Această componentă o vom numi *memorie permanentă*. (Astăzi, ne referim la această componentă prin *memorie externă*.)

5. Este necesară crearea posibilității de trecere de la memoria internă la cea permanentă şi reciproc.

Conform acestor principii, **arhitectura von Neumann** reprezintă structura ierarhică de organizare a unui calculator construit pe baza modelului von Neuman şi care conține: o unitate de memorie principală, o unitate centrală de prelucrare, o cale internă pentru transferul datelor şi dispozitive de intrare - ieşire.

Cu câteva excepții, şi calculatoarele actuale sunt construite pe baza acestor principii. Totuşi, **sistemele paralele**, numite şi arhitecturi non von Neumann, nu respectă principiul secvențialității (2), în timp ce **sistemele simbolice**, utilizate în aplicații complexe din domeniul inteligenței artificiale, se bazează pe inferențe logice, deci nu exclusiv pe calcule aritmetice (ca în principiul 1).

Opinia multor cercuri ştiințifice este că, în deceniul acesta, cele trei direcții fundamentale de cercetare care vor prezenta interes vor fi: microprocesoarele, inteligența artificială şi prelucrarea distribuită. Se preconizează astfel o orientare preponderent spre **arhitecturile non von Neumann**, mai noi şi mai puțin studiate…

1.1.9. Modelul formal al unui calculator

De la empirismul lui von Neumann, să trecem acum la rigoarea matematică. Incercăm să exprimăm formal sarcinile unui SC.

Notăm cu M, P, D, C cele patru componente principale: memoria, procesorul, dispozitivele periferice şi, respectiv, unitatea de control.

Principalele funcții ale SC le vom nota astfel:

r – *receive* – preluarea datelor de la un dispozitiv periferic şi depunerea lor într-o zonă de memorie;

t – *transmit* – este operația inversă lui *receive*;

s – *store* – preia o informație prelucrată de UCP şi o depune în memorie;

f – *fetch* – preia o dată din memorie şi o transmite componentei UCP spre prelucrare;

p – *processing* – prelucrarea propriu-zisă a datelor şi informațiilor.

Astfel, un SC poate fi exprimat prin:

SC = (M, P, D, C, r,t,s,f,p),

unde domeniile şi codomeniile celor cinci funcții sunt:

r : D x C —> M

t : M x C —> D

s : P x C —> M

f : M x C —> P

p : M x P x C —> M x P.

După cum se poate observa, fiecare funcție are, ca şi componentă a domeniului de definiție unitatea de control a SC.

1.1.10. Evoluția sistemelor de calcul

„De unde… un ENIAC este echipat cu 18000 de tuburi electronice şi cântăreşte 30 tone, înseamnă că computerele viitorului ar putea avea 1000 tuburi şi, probabil, o greutate de 1,5 tone. "

Popular Machine, martie 1949

Deşi este dificil de prezis performanțele viitoarelor SC, este sigur că vor depăşi cu mult caracteristicile celor actuale.

Atât creatorii de soft cât şi cei de hard construiesc sistemele de calcul bazându-se pe structura ierarhică de organizare în care fiecare nivel cumulează caracteristicile nivelelor inferioare.

Tehnologiile cheie pentru procesoarele moderne sunt compilatoarele şi siliconul. In timp ce siliconul permite mărirea capacităților hard, noile idei în organizarea calculatoarelor îmbunătățesc considerabil raportul preț / performanță. Dintre aceste idei, două sunt remarcabile: exploatarea paralelismului procesoarelor (de obicei prin pipeline) şi exploatarea accesului direct la memorie (de obicei prin completarea cu memorie de tip cache).

J. Presper Eckert şi John Mauchly de la Şcoala Moore a Univ. Pennsylvania construiesc prima maşină cu caracteristicile constructive şi funcționale ale unui calculator electronic operațional. Această maşină, numită **ENIAC (Electronic Numerical Integrator and Calculator)** a fost finanțată de SUA şi folosită în timpul celui de-al doilea război mondial, dar nu a fost făcută publică până în 1946. ENIAC era folosită în special pentru calcularea tabelelor atacului de artilerie. ENIAC avea 80 picioare lungime, 8,5 picioare înălțime şi câteva picioare adâncime. Fiecare dintre cei 20 regiştri de 10 biți

avea 2 picioare lungime. In total, ENIAC folosea 18000 tuburi electronice. (1m = 3,2808 ft)

ENIAC executa salturi condiționate şi era programabil. Programarea se făcea manual prin cuplarea (*plugging*) cablurilor şi setarea comutatoarelor, iar datele erau introduse pe cartele perforate (*punched cards*). Programarea calculelor de bază dura între o jumătate de oră şi o zi întreagă.

In 1944 John von Neumann a fost atras în proiectul ENIAC. Grupul vroia să îmbunătățească modalitatea de introducere a datelor şi se discuta asupra memorării programelor ca o succesiune de numere. Von Neumann a contribuit la cristalizarea ideilor şi a scris un memo prin care propunea un calculator cu program memorat (program-stored computer), numit **EDVAC** (**Electronic Discrete Variable Automatic Computer**). Acest memo este baza a ceea ce şi astăzi numim calculator von Neumann (vezi Principiile von Neumann). Mulți apreciază că această denumire acordă prea mare credit lui von Neumann şi neglijează aportul (inginerilor) creatorilor Eckert şi Mauchly…

In 1946 Maurice Wilkes de la Universitatea Cambridge a vizitat Scoala Moore. Intors la Cambridge, Wilkes s-a hotărât să demareze un proiect care să construiască un calculator cu program memorat, numit **EDSAC** (**Electronic Delay Storage Automatic Calculator**). EDSAC a devenit funcțional în 1949 şi se consideră primul calculator cu scopuri generale (engl., *general purpose calculator*), operațional şi cu program memorat din lume.

In timpul celui de-al doilea război mondial au fost construite calculatoare specializate pentru decodificarea mesajelor interceptate de englezi de la nemți. O echipă de la Bletchley Park, din care făcea parte şi Alan Turing, a construit în 1943 calculatorul Colossus. Această maşină a fost secretă până în 1970. După război, acest grup a avut o oarecare influență asupra pieței britanice de calculatoare.

In timp ce se lucra la ENIAC, Howard Aiken construia la Harvard un calculator electro-mecanic numit Mark-I. Acesta a fost urmat de Mark-II şi apoi de două maşini cu tuburi electronice, Mark-III şi Mark-IV. Față de EDSAC, care folosea o singură memorie atât pentru instrucțiuni, cât şi pentru date, Mark-III şi Mark-IV aveau memorii separate pentru instrucțiuni şi pentru date. Aceste maşini au fost considerate împotriva calculatoarelor cu program memorat. Pentru a respecta importanța maşinilor lui Howard, astăzi, termenul de **arhitectură Harvard** descrie maşini cu o memorie principală unică, dar cu zone cache separate pentru date şi instrucțiuni.

In 1947 a fost demarat la MIT (Massachusetts Institute of Technology) proiectul Whirlwind şi avea ca scop realizarea unor aplicații pentru procesarea în timp real a semnalelor radar. Deşi a condus la multe invenții, cea mai importantă rămâne memoria bazată pe miez (*core*) magnetic, care a fost baza sistemelor de memorare pentru următorii 30 de ani.

Firma IBM era implicată în afacerile cu cartele perforate şi cu automatizarea sistemelor, dar nu a construit calculatoare până în anii 1950. Primul calculator IBM, **IBM 701**, a fost scos pe piață în 1952 şi a fost vândut în 19 exemplare. Imediat după 1950 mulți erau pesimişti în legătură cu viitorul calculatoarelor, mai ales gândindu-se că piața şi oportunitatea pentru aceste maşini „super specializate" trebuie să fie foarte limitată.

In 1964, după o investiție de 5 miliarde de dolari, IBM a făcut un mare pas înainte când a anunțat calculatorul **System/360**. Acest sistem însemna o îmbunătățire a tuturor performanțelor de până atunci de cel puțin 25 de ori.

Un an mai târziu, DEC (Digital Equipment Corporation) a scos **PDP-8**, primul **minicalculator** comercial, la un preț sub 20.000 $. Minicalculatoarele au fost precursoarele miniprocesoarelor: în 1971 Intel a creat primul microprocesor, **Intel 4004**.

In 1963 s-a anunțat primul **supercalculator**, nu din partea companiilor mari sau a centrelor super-tehnologizate. Seymour Cray a creat în Minnesota maşina **CDC 6600** (CDC = Control Data Corporation). Această maşină includea multe din ideile regăsite în microprocesoarele de mai târziu.

Cray va părăsi ulterior CDC pentru a forma Cray Research Inc. în Wisconsin. In 1976 Cray Research Inc. lansează Cray-1 care era în acelaşi timp şi cea mai rapidă din lume, şi cea mai scumpă, dar şi calculatorul cu cel mai bun raport cost / performanță pentru programele ştiințifice. În 1996 Cray Research este asimilată de Silicon Graphics.

In timp ce Cray crea cel mai scump calculator, alții se gândeau cum să folosească microprocesorul pentru a crea un calculator atât de „scump" cât să poată fi cumpărat pentru acasă. Au fost mai multe tentative de a lansa un **calculator personal**, dar remarcabil rămâne faptul că în 1977 Apple II (i.e. Steve Jobs şi Steve Wozniak) a definit ce înseamnă: preț scăzut, volum mare şi calitate bună pentru ceea ce avea să devină industria calculatoarelor personale.

Deşi cu un avans de patru ani, Apple va sfârşi pe locul al doilea… Lansat în 1981, **IBM Personal Computer** devine cel mai bine vândut calculator de orice tip. Succesul lui va da câştig de cauză lui Intel pentru piața de microprocesoare şi lui Microsoft pentru sistemul de operare. Chiar şi astăzi, cel mai popular CD al lui Microsoft este cel cu sistemul de operare (Windows), chiar dacă este de multe ori mai scump decât un CD cu muzică!

Producția de microprocesoare este în plină ascensiune. Iată o situație la nivelul anului 1995:

Tip	Nr de buc vândute
80x86	50.000.000
MIPS	5.500.000
Power PC	3.300.000
Sun SPARC	700.000
HP PA-RISC	300.000
DEC Alpha	200.000

Dintre cele 5.500.000 microprocesoare MIPS (*Microprocessor without Interlocked Pipelined Stages*), numai 300.000 au fost folosite în SC. Celelalte au fost incorporate pentru jocuri video, imprimante laser ş.a.

Rezumat.

In acest paragraf am definit principalele concepte necesare pentru o bună înțelegere a structurii şi funcționării calculatorului. Dintre acestea, se remarcă prin importanță definiția pentru **arhitectura calculatoarelor**, pentru că acest concept dă şi titlul cursului.

Partea centrală a acestui prim paragraf din curs o constituie prezentarea structurii sistemului de calcul în diferite abordări: (1) tipurile de resurse ale calculatorului modern, (2) structura ierarhică de organizare a sistemului de calcul şi (3) structura fizică şi logică a sistemului de calcul.

Paragrafele următoare vor trata în detaliu tocmai componentele principale puse în evidență de structura logică a calculatorului, şi anume: sistemul de intrare/ieşire (dispozitivele periferice) şi unitatea de memorie, în capitolul întâi, structura şi rolul regiştrilor şi a UAL în capitolul al doilea, capitolul al treilea şi al patrulea fiind rezervate structurii şi funcționării unității centrale de prelucrare, concret a procesorului.

Cuvinte cheie.

sistem de calcul
hardware, software
arhitectura calculatoarelor, nivel conceptual de arhitectură
unitate funcțională, ecuație logică, UAL, UCP, memorie, IOS

Verificare.

1. Care sunt principalele dumneavoastră activități curente în care folosiți calculatorul?
2. Notați-vă principalele componente (denumire şi caracteristici) ale calculatorului pe care îl folosiți în mod frecvent. Decideți care sunt elementele pe care le cunoaşteți, pe care le puteți descrie şi care nu. Recitiți această descriere din când în când, pe măsură ce parcurgeți cursul şi bifați elementele despre care tocmai ați învățat. Discutați în final cu cadrul didactic ceea ce a rămas nebifat pe lista dumneavoastră.
3. Care este legătura între structura fizică şi structura logică ale sistemului de calcul?
4. Stabiliți-vă în scris câteva repere cronologice în evoluția sistemelor de calcul.

NOTIȚE

1.2. Sistemul de intrare/ieşire. Dispozitive periferice

Pentru a folosi SC, utilizatorul trebuie să introducă în calculator programele şi datele aplicației sale. De asemenea, după rezolvarea tuturor sarcinilor primite, SC trebuie să înştiințeze utilizatorul asupra rezultatelor obținute.

Totalitatea unităților funcționale capabile să *organizeze şi să memoreze informații externe*, pe de o parte şi să *asigure schimbul de informații între SC şi mediul extern*, pe de altă parte, alcătuiesc **sistemul de intrare/ieşire**. Concret, acesta cuprinde **dispozitivele periferice** sau, pe scurt, **perifericele**.

Principalele tipuri de dispozitive periferice legate la SC de tip IBM-PC sunt:

- Consola (**tastatura**) – prin intermediul căreia utilizatorul introduce datele în calculator foarte comod, sub formă alfanumerică;
- **Mouse**-ul – un dispozitiv de manevrare a unui cursor grafic care permite folosirea comodă a interfeței grafice;
- **Imprimanta** – cu rol în transmiterea informațiilor din calculator pe hârtie. La SC recente, imprimanta dispune de diferite seturi de caractere şi de puternice facilități grafice. **Exemple**: imprimante cu ace, cu jet de cerneală, cu laser.
- **Hard-discul** – ca principal dispozitiv de memorare a informațiilor externe;
- **Unitatea de disc flexibil** – un SC poate avea una sau mai multe unități de disc flexibil. La început s-au folosit dischetele de 8". Au urmat cele de 5.25" şi apoi unitățile de 3.5".
- **Unități de disc compact**, pentru citirea CD-urilor. Sistemele moderne au incorporate unități de disc compact *read-only*.

Pe lângă acestea, la SC se mai pot conecta şi alte periferice, întâlnite mai rar în practică. Ele sunt destinate unor categorii de utilizatori specializați în anumite operații. Dintre acestea amintim: plotter, scanner, unitate ZIP, overhead, CD-writer.

Parametrii care caracterizează performanțele unui periferic sunt:

1) modul de acces;
2) timpul de acces;
3) rata de transfer al informației;
4) capacitatea;
5) costul.

În funcție de tipul fiecărui dispozitiv periferic, unii parametri pot să nu aibă semnificație.

1.2.1. Clasificări

Din punctul de vedere al **direcției de transfer al informației**, avem: **periferice de intrare** (*input device*)**, periferice de ieşire** (*output device*) şi **periferice de intrare/ieşire** (*I/O device*). Din această ultimă categorie fac parte şi rețelele interne destinate comunicării între periferice.

Din punct de vedere **funcțional**, dispozitivele periferice se împart în:

1) **periferice de schimb**: imprimanta, plotter-ul, videoterminalele, rețelele de comunicare cu alte periferice;

2) **periferice purtătoare de informații permanent pe medii magnetice**: benzile magnetice, discurile (hard-discurile, dischetele).

Preluarea datelor de pe aceste suporturi se face prin intermediul unui cap de citire. Benzile magnetice sunt dispozitive cu acces secvențial la informațiile memorate. Capul de citire are poziție fixă şi el citeşte la un moment dat conținutul benzii care este în dreptul său.

La discurile magnetice, informația poate fi accesată în mod direct. Citirea se face prin deplasarea capului de citire până în dreptul zonei de pe disc unde este memorată informația căutată.

Observație. Discurile şi benzile magnetice pot fi discutate şi în contextul unităților de memorie auxiliară.

Din punctul de vedere al **tipului de transfer al informației**, avem: **periferice bloc** şi **periferice caracter**.

Perifericele bloc sunt caracterizate de faptul că organizează informația în blocuri de lungime fixă, fiecare bloc având propria adresă.

Unitatea logică de schimb dintre perifericele *bazate pe suport magnetic* şi memoria internă este **blocul de informație**. Acesta este format din unul sau mai multe sectoare vecine, care aparțin aceleiaşi piste. Un bloc va conține, pe lângă succesiunea propriu-zisă de octeți cu informație, un număr suplimentar de biți destinați verificării corectitudinii informației memorate în blocul respectiv. Pentru această completare sunt cunoscute două metode: schema de codificare polinomial-ciclică şi schema cu biți de paritate încrucişată.

Exemple: mediile de tip disc.

Perifericele caracter sunt caracterizate de faptul că furnizează sau primesc un *flux de octeți*, fără nici o structură de grupare a acestora. În consecință, octeții nu sunt adresabili şi fiecare octet este disponibil ca şi *caracter curent* până la apariția următorului caracter în/pe flux.

Exemple: imprimanta, terminalele cu tastatură şi ecran, mouse-urile.

Din punctul de vedere al **partenerului implicat în utilizarea perifericului respectiv**, avem: periferice care interacționează cu omul şi periferice care interacționează cu maşina.

O caracteristică importantă pentru aprecierea unui periferic este dată de **rata de transfer**. Aceasta reprezintă numărul de unități de informație pe

care le poate transfera perifericul respectiv într-o secundă în cadrul comunicării cu procesorul, cu memoriile sau cu alte periferice.

In tabelul următor enumerăm câteva dintre cele mai folosite periferice şi caracteristicile lor esențiale.

Device	Tip	Partener	Rata de transfer (KB / sec)
Tastatură	intrare	om	0,01
Mouse	intrare	om	0,02
Cititor de voce	intrare	om	0,02
Scanner	intrare	om	400
Redare voce	ieşire	om	0,6
Imprimantă pe linii	ieşire	om	1
Imprimantă laser	ieşire	om	200
Monitor	ieşire	om	60.000
Modem	intrare sau ieşire	maşină	2 - 8
Rețea LAN	intrare sau ieşire	maşină	500 - 6.000
Disk flexibil	memorare	maşină	100
Disk optic (CD)	memorare	maşină	1.000
Bandă magnetică	memorare	maşină	2.000
Disk magnetic	memorare	maşină	2.000 - 10.000

1.2.2. Caracterizarea unor periferice

1.2.2.1. Discurile dure. Hard-discul

Din punct de vedere fizic, un volum de disc magnetic are structura din figura următoare.

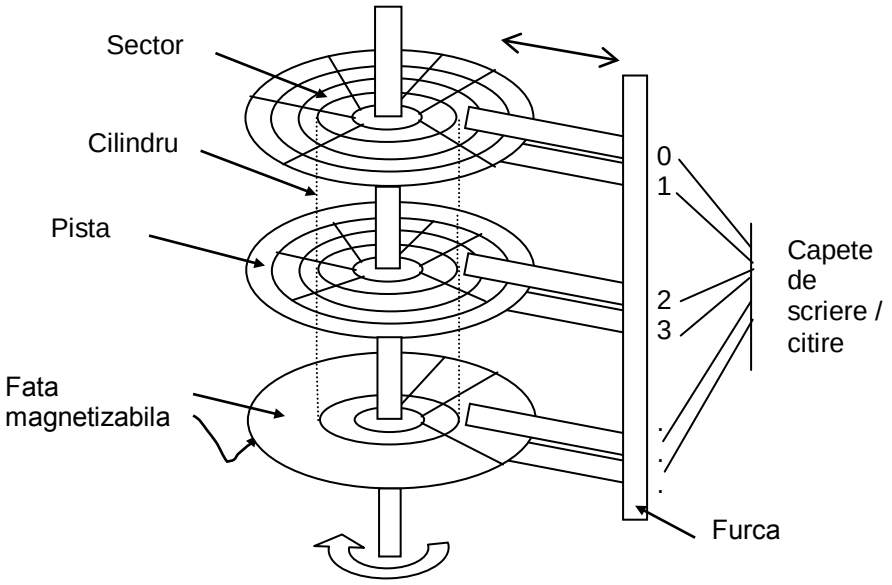
Volumul se montează pe o unitate de disc, unde se roteşte cu o viteză constantă. Suprafața de memorare a unui disc este structurată pe trei nivele (pistă, cilindru, sector) şi depinde de patru constante de construcție.

Suprafața fiecărei fețe active de memorare este divizată logic în coroane circulare concentrice numite **piste**. Numărul de piste de pe o față este prima constantă de construcție şi ea variază între 30 şi 800 piste/față.

In cazul în care volumul conține mai mult de două fețe active, situație frecvent întâlnită în practică, atunci fiecare față activă are acelaşi număr de piste şi toate pistele cu aceeaşi rază formează un **cilindru**. Numărul de fețe active ale unui volum de disc este a doua constantă de construcție şi ea variază între 1 şi zeci de fețe active/volum.

Fiecare pistă este împărțită în mai multe **sectoare**. Numărul de sectoare pe pistă este a treia constantă de construcție şi ea variază între 4 şi câteva zeci de sectoare/pistă. *Sectorul este unitatea de adresare a informației pe disc*.

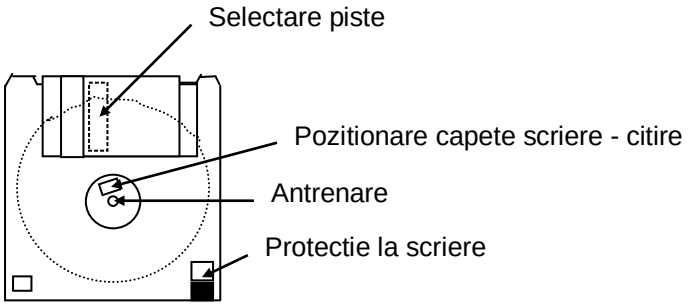
În fine, numărul de octeți dintr-un sector este a patra constantă de construcție. De regulă, acest număr este o putere a lui 2 şi variază între 128 şi 4096 octeți/sector.



Hard-discul este creat după anul 1985 şi rivalizează cu oricare dintre tipurile de discuri cunoscute. Capacitatea lor de memorare depăşeşte capacitatea de memorare a altor discuri.

1.2.2.2. Discurile flexibile. Discheta

Un *disc flexibil* este format dintr-o folie magnetizabilă pe ambele fețe, îmbrăcată complet în material plastic nedeformabil (la discurile de 3.5"). Principial, îmbrăcămintea are prevăzute patru obturații (vezi figura următoare): una centrală pentru antrenare, una radială pentru selectarea pistelor şi un orificiu în apropierea celui de antrenare, destinat poziționării capetelor de scriere - citire.



Al patrulea orificiu este în partea dreaptă jos: dischetele de 3.5" dispun (în vederea protecției la scriere) de un comutator cu două poziții: protejat la scriere, respectiv neprotejat. Astfel, dacă orificiul este închis atunci este permisă scrierea pe dischetă, iar dacă orificiul este deschis, se interzice scrierea.

Pentru o dischetă de 3.5” formatată la 1.44MB constantele de construcție sunt: 80 piste, 18 sectoare pe pistă, 2 capete de scriere-citire, 300 rotații pe minut, rată de transfer de 500Kbps (kilobiți pe secundă).

1.2.2.3. Compact discurile

Pentru gestionarea unor baze de date mari, casele de software utilizează din ce în ce mai mult discuri compacte, numite CD-ROM, ale căror capacități depăşesc 1000 MO. In general, informațiile de pe CD-ROM-uri sunt de tip *read-only* şi conțin de regulă baze mari de date cu caracter de consultare publică. Pe lângă CD-urile read-only, încep să se răspândească pe piață şi CD-urile reinscriptibile.

Primele astfel de discuri conțineau baze de date documentare pe diverse domenii ştiințifice: informatică, electronică, matematică, enciclopedii, etc.

1.2.2.4. **Monitoarele** (*graphic display*)

Orice SC are un monitor (ecran) prin care comunică cu utilizatorii.

Funcționarea monitorului se bazează pe tehnologia de realizare a televizoarelor, adică este folosit un tub catodic (CRT - cathode ray tube). Astfel, un fascicul de raze scanează imaginea linie cu linie, de 30 - 75 ori pe secundă. La această rată de scanare utilizatorul nu poate observa raza pe ecran.

Imaginea poate fi interpretată ca o matrice de elemente luminoase elementare (*picture elements*), numite **pixeli**. De aceea, imaginea poate fi reprezentată printr-o matrice de biți, numită **bit map**. Dimensiunile acestei matrice dau **rezoluția ecranului**. Cele mai simple monitoare folosesc câte un bit pentru reprezentarea unui pixel şi, astfel, pot reda numai imagini alb-negru. Urmează monitoarele care folosesc 8 biți pentru un pixel, care pot reprezenta astfel 256 nuanțe de gri ($256 = 2^8$). In fine, monitoarele color folosesc 24 biți pentru reprezentarea unui pixel şi 8 biți pentru fiecare dintre culorile fundamentale, adică roşu, verde şi albastru (RGB - Red, Green, Blue).

In particular, calculatoarele portabile au ecrane cu cristale lichide (LCD - liquid crystal display). Diferența esențială este faptul că un pixel LCD nu este o sursă de lumină.

Indiferent de tipul de monitor, componenta hardware pentru grafică conține mai multe buffere de imagine (*raster refresh buffer* sau *frame buffer*) pentru a memora imaginea sub formă de *bit map*. Imaginea de reprezentat pe ecran este memorată în *frame buffer* şi, conform cu rata de actualizare a imaginii (*refresh rate*), fiecare bit al fiecărui pixel este citit, interpretat şi reafişat pe ecran conform cu noile caracteristici (valori). Scopul realizării unui *bit map* este de a reprezenta fidel imaginea.

Competiția între sistemele grafice se dă pentru că ochiul uman detectează foarte bine fiecare schimbare de pe ecran. De exemplu, este neplăcut să se observe diferența dintre porțiunea de ecran pe care imaginea a fost actualizată, față de cea neactualizată.

Un monitor poate lucra în **regim text** sau în **regim grafic**.

Pentru conectarea monitorului la calculator se folosesc două modalități: legarea serială sau legarea prin adaptor video.

A. Legarea serială

Standardizat sub numele RS-232, acest mod de legare permite atât legarea locală cât şi legarea la distanță (geografic vorbind).

Transmisia se efectuează bit după bit. Pentru fiecare bit transmis se verifică un aşa-numit *protocol de comunicație*, în scopul asigurării corectitudinii transmiterii. Viteza de transmisie variază de la 300 bps (biți pe secundă) la 19.200 bps.

Atât tastatura, cât şi ecranul se supun acestui mod de conectare.

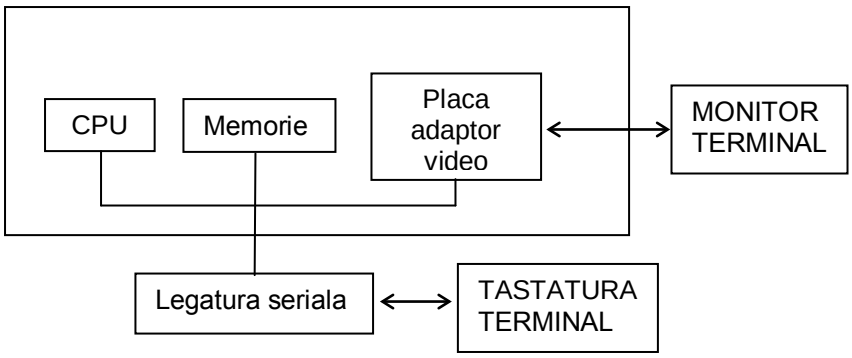
B. Legarea prin adaptor video

A doua mare categorie de monitoare sunt cele pentru care ecranul este legat la memoria calculatorului printr-o placă numită **adaptor video**.

Acest adaptor înglobează în el un **video-controller** care generează continuu un flux de electroni. La atingerea ecranului, aceştia se transformă în puncte luminoase. De aceea, la legarea prin adaptor video, ecranul este parcurs permanent linie cu linie (fenomenul se numeşte **baleiere ecran**), activându-se fiecare punct luminos în conformitate cu informațiile existente într-o porțiune a memoriei operative numită **memoria ecran**. Actualizarea ecranului în această manieră se desfăşoară în ritmul de 45-70 baleieri pe secundă.

Spre deosebire de legarea serială, aici o modificare a imaginii este un efect al modificării memoriei ecran. Acest lucru elimină necesitatea verificării prin protocoale a corectitudinii datelor, ceea ce face să crească mult viteza de afişare.

Acest sistem de lucru, ilustrat în figura următoare, este cunoscut şi sub numele de **legarea terminalelor cu memoria mapată**.



Ecranul poate lucra, dar nu simultan, fie în regim text, fie în regim grafic. În regim text vede (de regulă) 25 de linii a câte 80 de caractere fiecare, iar în modul grafic o matrice de pixeli, ale cărei dimensiuni depind de caracteristicile adaptorului video.

Evident că, pentru modul de lucru grafic memoria ecran este mult mai mare decât cea necesară pentru modul de lucru text.

In memoria ecran, fiecărui caracter (în mod de lucru text) sau fiecărui pixel (în mod de lucru grafic), îi sunt ataşate aşa numitele **atribute ale imaginii**. Aceste atribute se referă la:

- ✓ **strălucirea** punctului luminos (*brightness*);
- ✓ la posibilitatea imaginii de **a pulsa** pe ecran (*flashing)*;
- ✓ la posibilitatea imaginii de a apărea în **video invers** (adică situația în care se schimbă culoarea fondului cu cea a punctelor luminoase);
- ✓ la **culoarea** pixelului sau a caracterului.

1.2.2.5. Principalele adaptoare video

Primul adaptor video disponibil pentru IBM-PC a fost unul pentru ecran monocrom. El s-a numit **MDA** (*Monochrom Display Adapter*) şi a fost creat în 1981. Caracteristici: nu oferă facilități grafice, poate să afişeze numai caracterele ASCII standard şi cele speciale din setul IBM.

Adaptorul **Hercules,** prescurtat **HGC**, (*Hercules Graphics Card*) a fost creat de firma cu acelaşi nume în 1982. HGC permite atât afişare alfanumerică cât şi afişare grafică. Tehnicienii de la Hercules au rezolvat cu această ocazie îmbinarea textului cu grafica, iar standardul firmei s-a impus rapid pe piață. HGC permite comutarea între modul de lucru impus de MDA şi modul grafic cu o rezoluție înaltă: 720*348 pixeli.

Adaptorul **CGA** (*Color Graphics Adapter*) a început să fie livrat în 1981. Pe lîngă modul de lucru alfanumeric (80*25 caractere), acesta permite un mod de lucru cu patru culori şi o rezoluție de 320*200 pixeli. Calitatea afişajului nu este deosebită, datorită rezoluției destul de reduse, în schimb se pot folosi culorile.

Placa adaptor **EGA** (*Enhanced Graphics Adapter*) a apărut pe piață la începutul anului 1985. Ea moşteneşte modurile de lucru de la MDA şi CGA, dar oferă posibilități grafice mult mai mari. Astfel, pot fi utilizate simultan 16 culori din 64 posibile, cu o rezoluție de 640*350 pixeli. Evident, calitatea imaginii este mult mai bună decât cea oferită de adaptoarele precedente.

Interfața **VGA** (*Video Graphics Array*), apărută în 1987, aparține unui nou standard grafic. Este compatibilă cu plăcile anterioare, dar oferă posibilități mult sporite față de acestea. În mod grafic are o rezoluție de 640*480 pixeli cu 16 culori simultane din 262.144 posibile, sau 256 culori simultane cu 320*200 pixeli. În modul text caracterele sunt formate din 144 de pixeli (matrice de 9*16 pixeli pe caracter), față de 64 la CGA, 112 la EGA, 126 la MDA şi HGC.

În momentul de față există mai multe variante îmbunătățite de plăci VGA, cunoscute generic sub numele de **SVGA** (*Super VGA*).

1.2.2.6. Mouse-ul

Prin "mouse" ne referim la perifericul de pointare (*pointing device*) inventat în 1967 şi care a intrat în componența unui sistem de calcul de prin anii '80.

Principiul de funcționare se bazează pe utilizarea unei bile montată astfel încât să fie în contact cu doi cilindri de dimensiuni foarte mici. Unul dintre aceşti cilindri corespunde axei Ox, adică direcției orizontale, în timp ce celălalt corespunde axei Oy, adică direcției verticale.

Cilindrii sunt manevrați mecanic sau (la modelele moderne) antrenează câte o rotiță dințată prin intermediul căreia un LED (Light Emitting Diode) luminează un senzor foto.

Mişcarea fizică a mouse-ului pe *pad* determină învârtirea bilei, care determină rotirea cilindrilor Ox şi Oy (după cum mouse-ul este mişcat orizontal, vertical sau pe diagonală) şi, în continuare, actualizarea prin incrementare şi/sau decrementare a unor contoare sistem. Aceste contoare (numărătoare) au rolul de a înregistra cât de departe a fost mişcat mouse-ul şi în ce direcție.

Interfața între mouse şi sistem este asigurată prin una dintre variantele:

1. mişcarea mouse-ului generează impulsuri folosind LED-ul (vezi anterior) sau
2. mişcarea mouse-ului generează operații de actualizare a valorilor unor contoare.

Periodic, procesorul semnalează impulsurile sau citeşte valorile contoarelor şi determină deplasarea relativă a mouse-ului de la citirea anterioară. Conform cu rezultatul obținut, reprezentarea mouse-ului pe ecran este mutată proporțional. Mişcarea reprezentării mouse-ului pe ecran este lentă deoarece variația poziției mouse-ului este mai mică decât rata de citire şi interpretare a procesorului.

Mouse-ul este prevăzut cu două sau trei butoane. Operația esențială în lucrul cu mouse-ul este cea de **eliberare** a unuia dintre butoane.

Legătura dintre valorile contoarelor, poziția (starea) butoanelor şi poziția reprezentării mouse-ului pe ecran se face prin mecanisme software. De obicei, aceste programe sunt grupate în *driver*-ul de mouse şi se cumpără împreună cu perifericul. Deoarece există acest soft, utilizatorul poate seta diverşi parametri de lucru cu mouse-ul: intervalul de timp caracteristic unei operații *double-click*, viteza de deplasare a reprezentării mouse-ului pe ecran, s.a.

Pe de altă parte, interpretarea permanentă a poziției mouse-ului pe ecran asigură că niciodată mouse-ul „nu va sări din ecran". Metoda prin care sistemul primeşte informații despre mouse prin citirea şi interpretarea semnalelor de la mouse se numeşte *polling* [pouliη].

1.2.2.7. Rețele de comunicare

Dacă luăm în discuție comunicarea prin rețele atunci distingem două tipuri de rețele de comunicare: rețeaua internă a sistemului de calcul şi rețelele de comunicare între sisteme de calcul.

Rețeaua internă a unui SC interconectează componentele fizice interne ale sistemului, concret, procesorul la memorii şi la dispozitivele periferice.

Rețelele de comunicare între mai multe SC rezultă în urma interconectării componentelor a diferite calculatoare (sisteme multicalculator). Principalele avantaje ale realizării şi utilizării unei rețele de calculatoare sunt: comunicarea rapidă şi facilă, posibilitatea partajării resurselor, accesul la distanță.

O rețea de calculatoare se caracterizează prin:

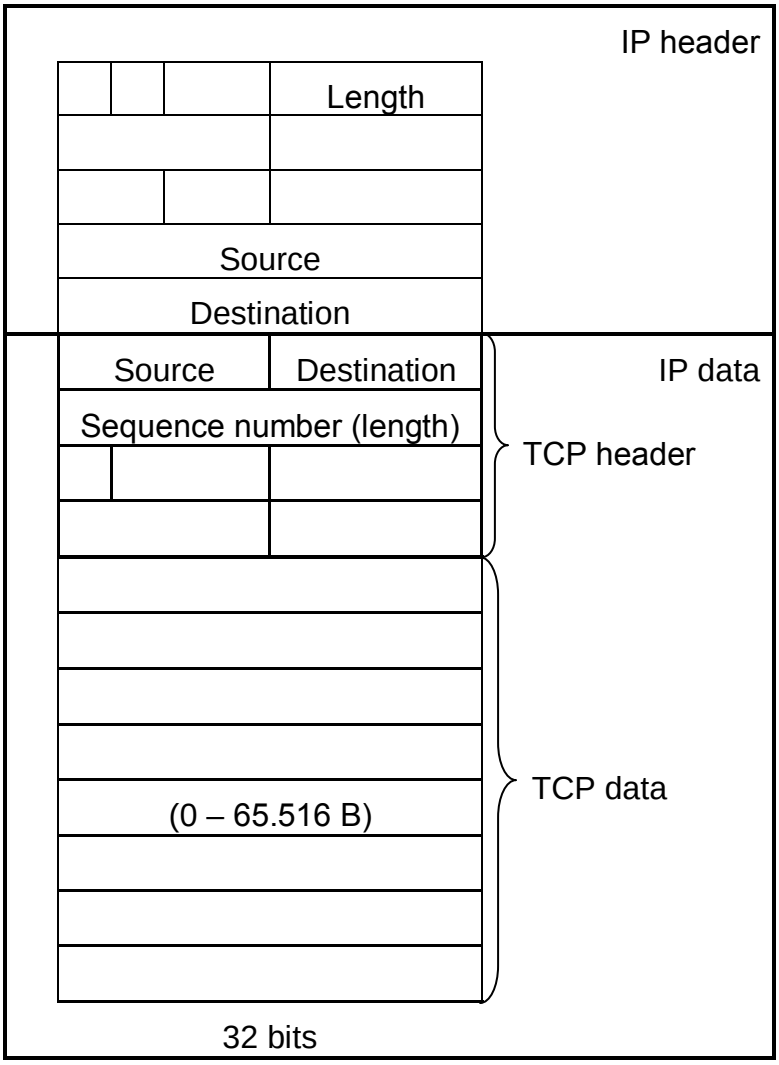
- ✓ distanța între SC conectate: de la 10 m la mii de km;
- ✓ viteza de transfer a mesajelor: de la 1 Byte pe secundă (Bps) la sute de MBps;
- ✓ topologia rețelei (organizarea SC conectate): bus, ring, star, tree etc.
- ✓ liniile de comunicare: fără linii de comunicare (rețele de tip point-to-point) sau rețele cu partajarea liniilor de comunicare (rețele multidrop).

Modele de rețele:

1) LAN (Local Area Network), cu exemplul cel mai popular, rețeaua **Ethernet**. Aceasta poate acoperi maxim 1 km, deci este potrivită pentru conectarea SC de la un etaj de clădire. Acceptă o viteză de transfer de 10Mbps (Megabiți pe secundă), fără control central. Mesajele transmise printr-o rețea Ethernet (pachetele) sunt dirijate de-a lungul rețelei în blocuri (64B/bloc – 1518B/bloc). Variantele moderne ale modelului LAN–Ethernet sunt Fast Ethernet (transferă 100Mbps) şi Gigabit Ethernet.

2) Long-haul Network, cu varianta cea mai populară, **ARPANET** (Advanced Research Projects Agency of US Government). Modelul ARPANET acoperă între 10 şi 10.000 km, la un transfer de 56 Kb / sec. Este de tip point-to-point. Foloseşte linii dedicate, închiriate de la companiile de comunicații.

   ARPANET a fost precursor pentru **Internet**. Prin realizarea Internet s-a creat posibilitatea interconectării mai multor rețele de calculatoare. Mai mult, s-a standardizat această conectare la o singură familie de protocoale, şi anume TCP/IP (Transmission Control Protocol / Internet Protocol). Componenta IP a acestor protocoale furnizează posibilitatea de adresare între două gazde (*host*-uri) din Internet, dar *nu* garantează securitatea şi siguranța transmisiilor. Partea TCP înseamnă un protocol care poate garanta că toate pachetele sunt primite şi că nu există erori de transmitere.



Aceste două protocoale (IP şi TCP) lucrează împreună şi formează o stivă de protocoale (*protocol stack*) în care pachetele TCP sunt încapsulate în pachetele IP. Pentru a permite comunicarea între mai multe rețele de caracteristici diferite, TCP/IP defineşte un format standard de pachet: un pachet IP (care conține informațiile de adresare IP) încapsulează un pachet TCP (care conține atât informații de adresare interpretate de gazdă, cât şi datele care se comunică).

1.2.2.8. Placa de rețea

Placa de rețea este componenta fizică a unui sistem de calcul prin care acesta se poate conecta la o rețea de calculatoare. O placă de rețea funcționează ca interfață fizică între calculator şi cablul de rețea. Fiecare calculator din rețea (stație) şi server-ul rețelei vor avea instalată câte o placă de rețea (fizic, într-un slot pe placa de bază sau incorporată direct în placa de bază). Legătura fizică între calculator şi restul rețelei se stabileşte după ce cablul de rețea se conectează la portul plăcii de rețea.

Fiecare „participare" a calculatorului în rețea presupune un transfer de date între calculator şi placa de rețea, deci un proces de comunicare. In cazul plăcilor cu acces direct la memorie, **DMA** Board (**D**irect **M**emory **A**ccess),

calculatorul-stație alocă o parte din spațiul său de memorie pentru placa de rețea.

1.2.2.9. Modemul

Atunci când se pune problema ca două calculatoare să comunice prin intermediul unei linii telefonice este nevoie şi de un dispozitiv numit **modem**. Necesitatea modemului rezultă din faptul că liniile telefonice pot transporta numai semnale analogice (sunet), în timp ce calculatoarele comunică prin impulsuri digitale (semnale electronice). Un semnal digital este unul discret, care are una din două valori posibile: 0 sau 1. Un semnal analogic poate fi reprezentat printr-o curbă continuă, având un domeniu infinit de valori.

Astfel, modemul apare ca un **mo**dulator-**dem**odulator de semnale, care converteşte semnalele digitale în semnale analogice şi invers.

Un modem include o interfață de comunicare serială (RS-232, vezi anterior *Legarea serială*) şi o interfață pentru linia telefonică, RJ-11 (adică o priză de telefon cu patru fire).

Modemurile sunt disponibile atât ca modele interne, cât şi externe. Un **modem intern** este instalat într-un slot de extensie al calculatorului, la fel ca orice altă placă de interfață. Un **modem extern** este ca o cutie conectată la calculator prin intermediul unui cablu serial (RS-232), care face legătura între portul serial al calculatorului şi conectorul de interfață serială al modemului. In plus, indiferent de model, modemul foloseşte şi un cablu cu conector RJ-11C pentru a se conecta la priza telefonică de perete.

1.2.3. Canalul de intrare / ieşire

Dezvoltarea arhitecturii a condus la apariția, începând cu generația a doua de calculatoare a *canalului de intrare / ieşire*. Acesta este un procesor specializat pe operații de I/O care poate funcționa în paralel cu CPU. Pentru a fi lansat în execuție, canalul primeşte de la CPU o comandă de efectuare a unei operații de I/O. După lansare, cele două procesoare îşi continuă activitatea în paralel. La sfârşitul operației de I/O, canalul anunță acest fapt CPU prin intermediul unei *întreruperi*.

În prezent, SC folosesc două tipuri de canale: selector şi multiplexor.

Canalele de tip selector realizează schimbul dintre memorie şi dispozitivele rapide (discuri, benzi). Acestea lucrează la un moment dat cu un *singur periferic*.

Canalele multiplexor lucrează simultan cu *mai multe periferice*. Astfel, trebuie să decidă, pentru fiecare octet schimbat, care este traseul acestuia între memorie şi perifericele legate la el.

1.2.4. Legarea perifericelor la SC. Unitățile de interfață

Perifericele conectate la un SC au nevoie de linii speciale de comunicare pentru a se lega la UCP. Concret, aceasta se realizează prin componente hardware numite **unități de interfață** cu rol în supervizarea şi sincronizarea tuturor transferurilor de intrare / ieşire**.**

Principalele funcții ale unităților de interfață sunt:

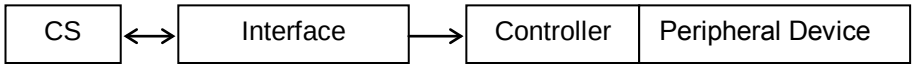
1) conversia semnalelor specifice modului de operare pe suporturi magnetice în semnale de lucru pe circuite electronice, adică de pe periferic magnetic în CPU;
2) sincronizarea ratelor de transfer de date de la viteza de lucru pe periferic la viteza de lucru în CPU;
3) decodificarea/codificarea datelor de pe periferic în format specific în CPU;
4) delimitarea semnalelor unui periferic de ale altora dintre cele conectate la acelaşi SC.

În funcție de modul de transmitere a informației spre periferic distingem **interfețe seriale** şi **interfețe paralele**. De obicei, un PC are o interfață paralelă (LPT1) pentru imprimantă şi două interfețe seriale (COM1 şi COM2) pentru tastatură şi mouse.

Comunicarea serială presupune că informația circulă într-o structură secvențială de biți în care alternează biții de informație cu biții de control. În comunicarea paralelă, informația circulă în blocuri de 8 biți. Avantajul comunicării paralele constă în mărirea vitezei de transfer şi în eliminarea biților de control.

Pe lângă interfață, fiecare periferic poate avea o unitate de control proprie, numită **controller**, cu rol în supervizarea operațiilor specifice mecanismului de funcționare a perifericului respectiv. De exemplu, controllerul propriu imprimantei gestionează deplasarea hârtiei, timpul de listare şi selectarea tipului de caracter de imprimat.

Controller-ul poate fi independent sau poate fi integrat fizic cu perifericul.



Punctul în care se conectează fizic interfața la SC este un registru de date care se numeşte **port**. Practic, portul este registrul prin care se realizează schimbul de informații între sistem şi exterior. În funcție de tipul de periferic legat printr-un port, acesta poate fi **port de intrare**, **port de ieşire** sau **port de intrare / ieşire**.

Portul de intrare / ieşire specifică un canal prin care informația circulă între dispozitivele hardware şi UCP. Portul este identificat de UCP printr-o adresă. Astfel, fiecare resursă hardware din sistem trebuie să aibă o altă adresă de port I/O.

Concret, comunicarea între procesor şi diferite periferice se face prin **magistrala de intrare / ieşire** (I/O Bus).

Din punct de vedere logic, legătura între dispozitivele periferice, memorii şi UCP este asigurată de o rutină de interfață care se va numi **driver**. Concret, un driver este o aplicație software care identifică perifericul în sistem.

Comunicarea între un periferic şi UCP este permanentă. In acest sens, procesorul citeşte şi interpretează periodic anumite date despre starea fiecărui periferic. Subsistemul de gestiune a perifericelor poate testa periodic

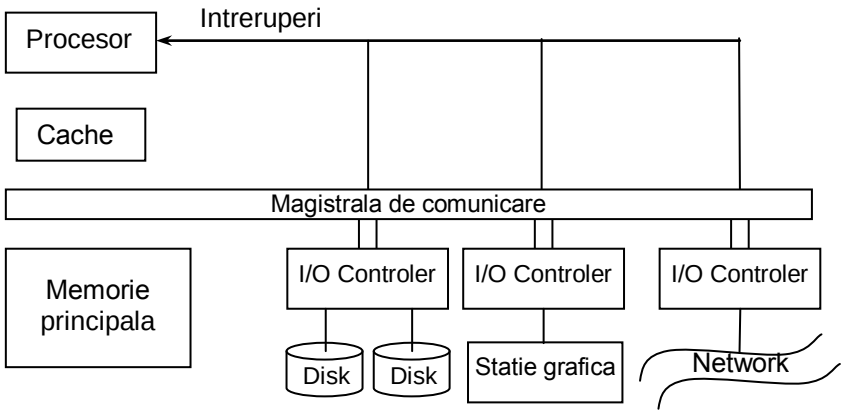
fiecare periferic la acelaşi interval de timp sau pe baza unui sistem de priorități. Tehnica „sondării" periodice de către procesor a stării perifericelor se numeşte **polling** [pouliη]. Testarea pe baza unui sistem de priorități este o metodă „de încredere" şi poate fi cea mai potrivită soluție în sistemele mici. Tehnica *polling* este o metodă nerecomandată în sistemele puternic dependente de timp (*time-critical*) deoarece un eveniment important poate apărea imediat ce perifericul său tocmai a fost testat; în acest caz evenimentul este lăsat în aşteptare până la următoarea testare, ceea ce nu este convenabil pentru un eveniment important. O alternativă modernă, deosebit de utilă în toate tipurile de sisteme, este folosirea întreruperilor.

1.2.5. Magistrale

O **magistrală** conectează dispozitivele de intrare - ieşire la procesor şi la memorii. O **magistrală** reprezintă o linie de comunicare partajată care foloseşte un set de conductori pentru a conecta mai multe subsisteme.

Magistrala reprezintă interconectarea electrică între periferice, procesoare şi memorii şi, deci, defineşte protocolul de comunicare la nivelul cel mai de jos (*lowest level*).

Un ansamblu tipic de periferice conectate la procesor şi memorii se poate reprezenta ca în figura următoare.



Folosirea magistralelor de comunicare are în principal avantaje legate de flexibilitatea conectării componentelor. Astfel, pentru o schemă deja definită se pot adăuga cu uşurință noi tipuri de periferice şi, mai mult, perifericele pot fi mutate de la un SC la altul, dacă acestea folosesc acelaşi tip de magistrală.

In cazul în care magistrala este suprasolicitată, apare dezavantajul major al comunicării „gâtuite" (engl., *bottlenecked*). Această situație poate fi evitată prin fixarea (şi respectarea) unui număr maxim de intrări (periferice conectate) pe magistrală.

Performanțele unei magistrale se apreciază din două puncte de vedere fundamentale: viteza maximă de transfer a datelor pe acea magistrală şi numărul maxim de periferice pe care le poate conecta. Scopul concret pentru care este construit fiecare model de magistrală va decide care dintre

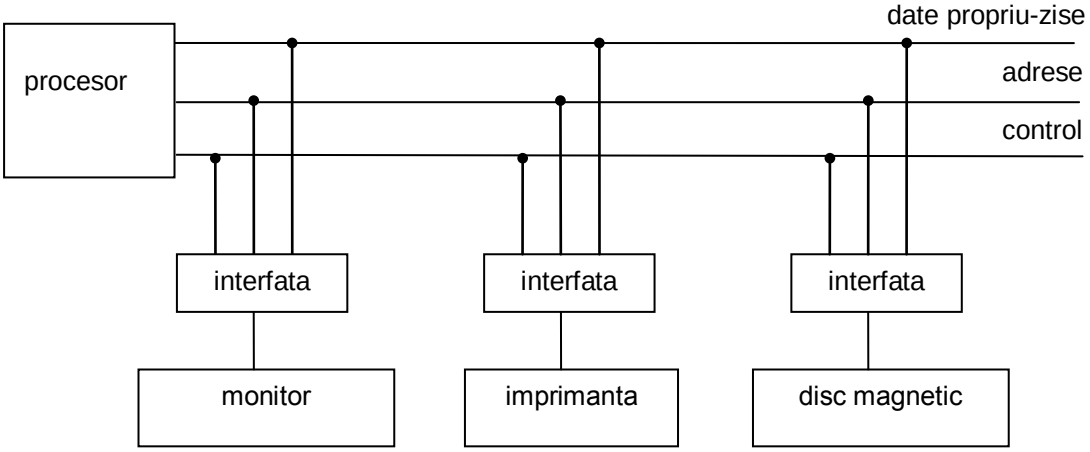
aceste criterii este prioritar şi, în consecință, va valorifica la maximum acea caracteristică, în detrimentul celeilalte. Proiectarea unui model este dificilă, ținând cont şi de faptul că viteza de transfer este limitată la rândul ei de factori fizici cum ar fi: numărul de periferice conectate şi lungimea magistralei.

In general, o magistrală constă în linii de date şi linii de control. Numărul liniilor de un anumit tip este proporțional cu viteza transferului pe liniile respective.

Pe liniile de date se transferă informații de la o entitate sursă la o entitate destinație. Aceste informații pot fi: date, comenzi sau adrese. Rolul liniilor de control este: să semnaleze cererile, să precizeze tipul datelor de pe liniile de date, să implementeze protocolul de magistrală. Deoarece magistrala este partajată, trebuie stabilit un protocol care să decidă univoc la fiecare moment cine foloseşte magistrala la momentul următor. Operația de bază cu care lucrează un protocol de magistrală este **tranzacția pe magistrală**, care constă din trimiterea adresei, pe de o parte şi transmiterea şi primirea datelor, pe de altă parte. O astfel de tranzacție este definită de direcția de lucru cu memoria, rezultând **tranzacții de intrare** şi **tranzacții de ieşire**.

Exemplu de utilizare a liniilor unei magistrale. Dacă un disc vrea să scrie date în memorie de pe un anumit sector al său atunci liniile de date se vor folosi pentru a preciza adresa din memorie la care se va face scrierea datelor şi, ulterior, tot pe liniile de date se vor transfera şi datele propriu-zise de pe disc. In acest caz, liniile de control vor indica ce tip de date (adresă sau date propriu-zise) sunt pe liniile de date la fiecare moment al tranzacției.

Observație. Unele magistrale au două seturi de linii de date pentru a separa datele de adrese. In continuare este descris un model de transfer pe o magistrală cu linii de date, de adresă şi de control.



Liniile de magistrală I/O care pornesc de la procesor sunt legate la unitățile de interfață ale tuturor perifericelor. Pentru a comunica cu un anumit periferic, procesorul pune adresa perifericului pe liniile de adresă. Fiecare interfață conectată la magistrala I/O conține un decodificator de adrese care gestionează liniile de adresă. Când o interfață detectează propria sa adresă, deschide accesul între liniile de magistrală şi perifericul pe care îl gestionează. Pentru toate interfețele cărora nu le corespunde adresa de pe magistrală, perifericele respective sunt inaccesibile.

Când procesorul pune adresa perifericului căutat pe liniile de adresă, în acelaşi timp furnizează pe liniile de control şi codul unei funcții (*function code*). Interfața selectată răspunde acestui cod şi determină execuția funcției. Codul acesta este o **comandă de intrare/ieşire** şi constă într-o instrucțiune de executat de către interfață şi perifericul asociat ei. Interpretarea comenzii depinde de perifericul căreia îi este adresată. O interfață de periferic poate primi următoarele patru tipuri de comenzi:

1) comenzi de control – care activează perifericul sau îl informează asupra următoarei operații;

2) comenzi de stare – cu rol în testarea stării interfeței şi a perifericului. Erorile care apar la transferul datelor sunt semnalate prin valorile anumitor biți din registrul de stare. Procesorul citeşte periodic acest registru, detectând astfel situația în care a apărut o eroare;

3) comenzi de ieşire – care constau în comenzi de transfer al datelor înspre exterior, adică determină interfața să răspundă prin dirijare de date de pe magistrală într-unul din regiştrii săi de tranzit (*buffer register*);

4) comenzi de intrare – care sunt opuse comenzilor de ieşire. În acest caz, interfața primeşte un set de date de la periferic, pe care le depune într-unul din regiştrii săi de tranzit. Procesorul verifică printr-o comandă de stare dacă datele sunt accesibile şi apoi transmite o comandă de intrare. Interfața depune datele pe liniile de date ale magistralei I/O de unde sunt preluate de procesor.

Clasificarea magistralelor.

Din punctul de vedere al componentelor conectate, avem:

1) magistrale procesor - memorie;

2) magistrale de intrare - ieşire;

3) magistrale de extensie (*backplane*).

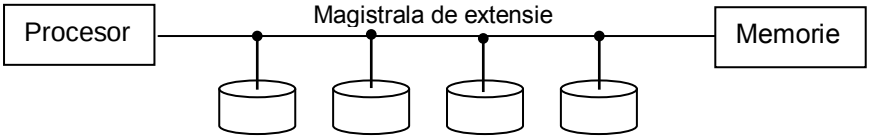
Magistralele procesor - memorie sunt, în general, scurte, dar de viteză mare şi adaptate sistemului de memorie, astfel încât să maximizeze lungimea de bandă procesor - memorie şi, implicit, cantitatea de informație transmisă simultan.

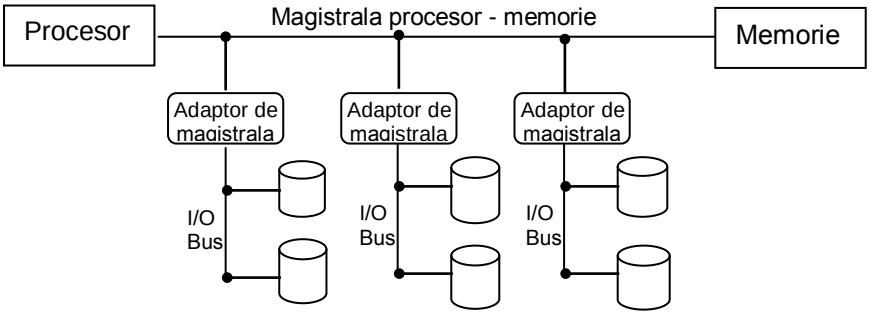
Magistralele de intrare - ieşire pot fi suficient de lungi, pot conecta mai multe tipuri de periferice şi acceptă o varietate mare de lungimi de bandă pentru perifericele conectate. Magistralele de intrare-ieşire nu interacționează direct cu memoria ci folosesc o magistrală procesor - memorie sau una de extensie pentru conexiunea cu memoria.

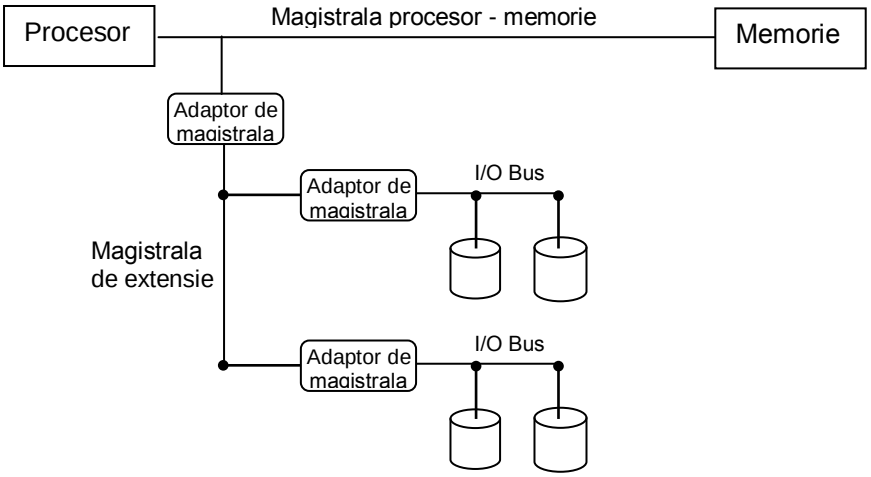
Magistralele de extensie (*backplane*) sunt construite pentru ca procesoarele, memoriile şi perifericele să poată coexista pe aceeaşi magistrală. Astfel, o magistrală de extensie va răspunde atât cererilor de comunicare procesor - memorie, cât şi cererilor de comunicare dintre periferice şi memorie. Se numesc de extensie tocmai pentru că permit adăugarea în sistem a diferitelor plăci interschimbabile, prin inserarea acestor plăci în conectorii de extensie ai magistralei (sloturi).

Comparativ, putem spune că magistralele procesor - memorie au design specific şi sunt, în general, monopol al firmei producătoare, în timp ce celelalte două tipuri (de intrare - ieşire şi de extensie) pot fi refolosite în diferite sisteme şi sunt construite pe baza unor **standarde de magistrală**.

In figurile următoare reprezentăm trei variante de interconectare a componenetelor SC prin magistrale de comunicare.







Avantajele remarcabile ale variantei a treia, cea mai complexă, ar fi:

- magistrala procesor - memorie poate fi mult mai rapidă decât magistralele de extensie sau de intrare - ieşire;
- sistemul de intrare - ieşire poate fi completat cu multe *controller*-e sau chiar cu magistrale I/O pe magistrala de extensie — ceea ce nu afectează viteza magistralei procesor - memorie.

Din punctul de vedere al tipului de comunicare, avem:

1) magistrale sincrone;

2) magistrale asincrone.

O **magistrală sincronă** are pe liniile de control un ceas care controlează un protocol bine definit pentru comunicare. De obicei, magistralele procesor - memorie sunt magistrale sincrone, deoarece conectează puține componente şi trebuie să lucreze la viteză mare de transfer a datelor.

Observație. Funcționarea unei magistrale sincrone poate fi simulată cu un automat finit determinist.

Prezența ceasului impune restricții în construirea şi funcționarea unei magistrale sincrone. De exemplu, toate componentele conectate trebuie să funcționeze la aceeaşi frecvență de tact. De asemenea, datorită problemelor de sincronizare, o magistrală sincronă nu poate fi şi lungă şi rapidă.

O **magistrală asincronă** nu este controlată de ceas, deci poate grupa o mai mare varietate de periferice şi poate fi oricât de lungă. In acest caz, pentru coordonarea transmiterii de date se foloseşte un protocol de tip *hand shaking* (vezi şi Moris Manno, pag. 393). Acesta constă dintr-o succesiune de paşi în care sursa şi destinatarul transferului de date trec la pasul următor numai de comun acord. Acest protocol este practic implementat cu suplimentarea liniilor de control pe magistrală.

Observație. Funcționarea unei magistrale asincrone poate fi simulată cu o pereche de automate finit deterministe: o maşină trece în starea următoare numai dacă cealaltă a intrat într-o anumită stare - test.

Standarde de magistrale.

Magistralele de intrare - ieşire servesc ca o modalitate de a îmbunătăți performațele SC prin conectarea a noi periferice. Pentru a facilita această operație, industria sistemelor de calcul a dezvoltat o serie de standarde de magistrală.

Un **standard** reprezintă o specificare pentru un producător de componentă. Astfel, un **standard de magistrală** impune producătorilor structura de bază a magistralei, astfel încât să poată fi cunoscută şi folosită a priori de producătorii perifericelor, cu compatibilitate maximă cu maşina gazdă.

Un standard de magistrală este, de exemplu, **SCSI** – *Small Computer System Interface*. De obicei, o magistrală de acest tip este o interfață fizică către o magistrală de extensie sau către o magistrală procesor - memorie. Un *controller* SCSI coordonează transferurile de la periferic, pe magistrala de intare - ieşire, spre memorie, prin intermediul unei magistrale procesor - memorie. Deci, magistralele de intrare - ieşire sunt construite conform acestui standard.

Pentru magistralele de extensie este recunoscut standardul **ISA** – *Industrial Standard Architecture*. Inițial au fost magistrale pe 8 biți şi, ulterior, au fost extinse la 16 biți, ca standard pentru IBM PC/AT. Limitată la o frecvență de ceas de 8 MHz, această magistrală a fost deja insuficientă pentru procesoarele 80386. Totuşi, o magistrală ISA (sloturi de 8 biți sau de 16 biți) rămâne o opțiune pentru conectarea imprimantelor prin porturi seriale,

pentru *controller*-e de mouse, dar sunt prea lente pentru discurile moderne sau pentru adaptoare video.

Standardul **PCI** – *Peripheral Component Interconnect* – a fost inițiat de Intel şi dezvoltat ulterior de alt concern industrial. Aceste magistrale sunt cele de mare viteză. PCI a fost prima magistrală care a răspuns cererilor de magistrală pe 32 biți reclamate de procesoarele Pentium şi opera la frecvențe de tact de 33 MHz sau 66 MHz, permițând rate de transfer de 132 MB / sec sau 264 MB / sec. PCI suportă controlul total al magistralei (*bus mastering*). De obicei, magistralele de extensie sunt magistrale PCI. Arhitectura magistralei PCI poate oferi facilitatea *Plug-and-Play* (autoconfigurare). Această tehnologie constă în adaptarea automată a configurației calculatorului (fără intervenția utilizatorului) la caracteristicile componentei care tocmai se instalează. Astfel, instalarea oricărui dispozitiv devine o operație simplă şi sigură. In plus, autoconfigurarea face inutilă setarea manuală a configurației calculatorului. Sistemul de operare Windows 95 (şi ulterioarele) respectă specificațiile *Plug-and-Play*.

Calculatoarele personale dispun de patru tipuri de arhitecturi de magistrală şi anume: ISA, EISA (Extended ISA), Micro Channel şi PCI.

Pentru conectarea dispozitivelor periferice de viteză redusă la calculator s-a impus un standard de magistrală unanim acceptat, anume **USB** (engl. *Universal Serial Bus*).

1.2.6. Transferul eficient de date. Procesoare specializate

Funcția principală a unei interfețe de intrare / ieşire este transferul de date. In funcție de gradul de implicare a procesorului în controlul transferului de date, avem:

1. transfer prin **program** – în care UCP se implică total în efectuarea transferului. Un astfel de program de control al transferului este o rutină de intrare / ieşire. Pe parcursul transferului de date procesorul trebuie să se adapteze la viteza de lucru a perifericului (în general mult mai mică decât a procesorului);
2. transfer prin **întreruperi** – în care UCP poate să folosească timpul liber dintre două transferuri consecutive pentru executarea altor secvențe de program (execuție concurentă); tratarea întreruperilor va fi reluată în capitolul al treilea;
3. transfer prin **acces direct la memorie** (DMA, engl. *Direct Memory Access*), fără intervenția efectivă a procesorului; această metodă de transfer necesită un circuit specializat (numit controller DMA) care are posibilitatea de a prelua controlul magistralei sistem în vederea efectuării transferului propriu-zis de date;
4. transfer prin **procesor de intrare/ieşire** – care conferă procesorului cel mai mare grad de libertate deoarece sarcinile legate de transfer sunt preluate de către o altă unitate de tip procesor, numită procesor de intrare / ieşire.

Alegerea modului optim de transfer pentru o anumită interfață depinde de o serie de factori, printre care: viteza de transfer a perifericului la care se

conectează, complexitatea perifericului respectiv, restricțiile de cost şi timp de realizare a interfeței respective. Implicit, structura şi modul de funcționare a interfeței depind de modul de transfer adoptat.

Situația descrisă în paragrafele anterioare, în care fiecare interfață comunică cu UCP, poate fi înlocuită cu aceea în care un SC are incorporate unul sau mai multe procesoare externe cărora le încredințează sarcina de a comunica direct cu dispozitivele periferice ataşate sistemului. Un astfel de procesor se va numi **procesor de intrare/ieşire, IOP**.

Ca un caz particular, un procesor care comunică cu terminale aflate la distanță (*remote terminals*) prin telefon sau alte medii de comunicare serială se va numi **DCP, data communication processor**. El este specializat pentru distribuirea şi colectarea datelor direct de pe rețele de comunicații. O rețea de comunicație este compusă din dispozitive periferice de diferite tipuri: imprimante, videoterminale interactive, senzori digitali şi/sau unități de intermediere a comunicării la distanță.

Folosind un DCP, SC poate rezolva cereri diferite din rețea de o manieră partajată, astfel că se poate comporta ca şi când ar deservi mai mulți utilizatori simultan. Aceasta este maniera în care SC este capabil să opereze eficient în *timp partajat* (*time - sharing*).

Principala diferență între IOP şi DCP constă în felul în care procesorul comunică cu perifericele. Un IOP comunică cu dispozitivele I/O printr-o magistrală I/O comună. Astfel, toate perifericele partajează magistrala comună şi o folosesc pentru transferul de date înspre/dinspre IOP. Un DCP comunică cu fiecare periferic în parte printr-o singură pereche de conductori (*a single pair of wires*). Atât datele cât şi informațiile de control sunt transferate de o manieră serială, astfel că rata de transfer este mult încetinită. Sarcina unui DCP este să transmită şi să colecteze informații digitale înspre/ dinspre fiecare terminal. Totodată, trebuie să determine dacă informația este de tip dată sau este informație de control şi să răspundă tuturor cererilor stabilite anterior prin proceduri de funcționare.

Terminalele aflate la distanță sunt conectate la DCP prin linii telefonice sau prin alte modalități de comunicare publice sau închiriate. Deoarece liniile telefonice au fost la origine destinate comunicării prin voce şi SC comunică prin semnale digitale, trebuie folosită o modalitate de conversie între cele două tipuri de semnale. Aceste convertoare se numesc: *data sets, acoustic couplers* sau *modems*.

Rezumat.

Scopul acestui paragraf a fost prezentarea caracteristicilor esențiale ale principalelor periferice conectate la un calculator modern, adică periferice standard (tastatură, mouse, monitor) şi periferice larg utilizate (discuri, imprimantă, modem, plăci de conexiune). Tratarea acestor periferice s-a făcut din mai multe puncte de vedere: (1) identificarea diferitelor tipuri de periferice (clasificări), (2) caracterizarea elementelor specifice fiecărui tip de periferic, (3) caracterizarea perifericelor din perspectiva conectării lor cu componentele cu care comunică direct.

Cuvinte cheie.

dispozitiv periferic, sistemul de intrare/ieşire
periferic de intrare, periferic de ieşire, periferic de intrare/ieşire
periferic de schimb, periferic de memorare
unitate de interfață, port
magistrală, standard de magistrală

Verificare.

1. Care sunt perifericele conectate la sistemul de calcul pe care îl folosiți în mod frecvent?
2. Explicați modul în care procesorul central stabileşte comunicarea directă cu un anumit periferic solicitat la un moment dat.

NOTIȚE

1.3. Unitatea de memorie

Unitatea de memorie este o colecție de componente destinate memorării datelor şi de circuite asociate lor, necesare transferului de informații înspre şi dinspre memorie.

1.3.1. Structura fizică a memoriei

Referitor la ***structura fizică*** a memoriei, avem că **memoria** este formată dintr-un mediu de memorare **şi** sisteme de circuite electromagnetice pentru controlul gestiunii memoriei. La rândul său, din 1919 subsistemul de memorare este alcătuit dintr-o succesiune de **bistabili** (circuite basculante bistabile) ce permit sau nu trecerea curentului, fiecare stabilind astfel memorarea uneia dintre valorile 0 sau 1.

Observație*.* Reprezentarea a două stări posibile ale unui sistem prin 0 şi 1 a determinat folosirea în sistemele de calcul în special a bazei de numerație 2 şi a puterilor lui 2.

In general, cantitatea de informație care se câştigă prin precizarea stării unui sistem care are două stări posibile se numeşte **bit de informație.** In particular pentru memorii, vom înțelege prin **bit** informația referitoare la starea unui bistabil: 0 sau 1. Cu alte cuvinte, un bistabil îndeplineşte funcția de memorare a unui bit de informație.

In memorie, un grup de bistabili se numeşte **locație de memorie**. Pentru SC actuale, o locație are opt biți şi se numeşte **octet** (O)**,** sau **byte** (B)**,** sau **caracter**. Această echivalență de denumire provine de la faptul că un caracter se reprezintă în memorie pe un octet.

Octeții sunt numerotați de la 0 la capacitatea maximă a memoriei. Astfel, capacitatea memoriei este dată de cantitatea de informație pe care aceasta o poate stoca, adică numărul de octeți dați eventual, multiplicativ: 1 KB (kilobyte) = 2^{10}B = 1024 octeți, 1 MB (megabyte) = 2^{10} KB = 2^{20} octeți, 1GB (gigabyte) = 2^{10} MB = 2^{30} octeți ($2^{30} = (2^{10})^3 \approx (10^3)^3 = 10^9$), 1TB (terrabyte) = 2^{10} GB = 2^{40} octeți.

Numărul octetului reprezintă **adresa de memorie** a grupului respectiv. Această adresă joacă un rol esențial în regăsirea informației memorate şi transmiterea ei către alte subsisteme. **De exemplu**, dacă unitatea centrală cere transmiterea unei informații memorate atunci ea trebuie să furnizeze atât *adresa octetului de unde se face transferul, cât şi numărul de octeți care se transferă*.

In terminologia modernă, numim **cuvânt** orice succesiune de 2, 4 sau 8 octeți. Astfel, calculatoarele din primele generații, până la 80286 inclusiv, au cuvântul format din 2 octeți, începând cu seria 80386, cuvântul este de 4 octeți, iar la microcalculatoarele moderne avem cuvântul de 32 biți = 4 octeți sau 64 biți = 8 octeți. Supercalculatoarele au cuvântul format din 8 octeți. Lungimea cuvântului este o caracteristică a procesorului.

Biții unui cuvânt trebuiesc ordonați într-un mod unic. Pentru aceasta se foloseşte ordinea puterilor lui 2. Toate tipurile de SC numerotează biții începând cu 0. Dintre biții unui octet se remarcă:

- **bitul cel mai semnificativ, MSB,** cel care corespunde puterii celei mai mari a lui 2;
- **bitul cel mai puțin semnificativ, LSB,** cel care corespunde puterii celei mai mici a lui 2.

In funcție de modelul de procesor, biții unui octet sunt numerotați crescător de la stânga la dreapta (format *big endian*, specific procesoarelor Motorola) sau de la dreapta la stânga (format *little endian*, specific procesoarelor Intel).

Tehnologia circuitelor integrate s-a impus cu succes şi în construcția memoriilor, astfel încât spunem despre un ansamblu de bistabili dispuşi într-o ordine bine precizată că formează un cip de memorie. Cele mai răspândite modele de cipuri sunt de tip SIMM (engl. *Single Inline Memory Module*) sau DIMM (engl. *Dual Inline Memory Module*). O placă de memorie SIMM sau DIMM este un grup de cipuri de memorie cablate pe o placă de dimensiuni reduse şi vândute ca o singură unitate (memorie). Deosebirea dintre SIMM şi DIMM constă în felul în care este plasată seria de conectori (pini de contact) prin care memoria respectivă se conectează pe placa de bază şi anume dispunere pe o singură parte sau pe ambele părți ale plăcii de memorie. Fiecare conector are o funcție bine delimitată în transferul de date memorate, în adresare sau în controlul memoriei respective. O configurație tipică de SIMM este cu 8 cipuri de 32Mb (megabiți) fiecare, ceea ce înseamnă 32MB (megabytes) de memorie.

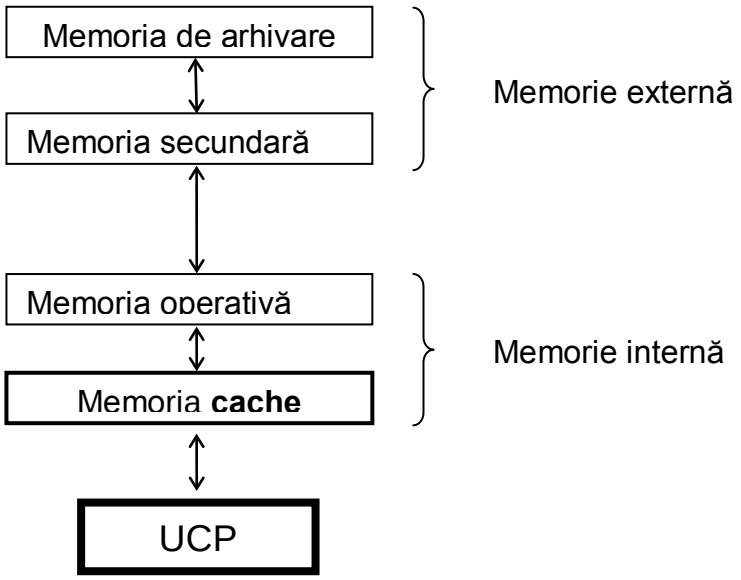
1.3.2. Structura ierarhică de organizare a memoriei

In timpul execuției lor, programele accesează în orice moment o zonă relativ mică a spațiului de adrese. Această observație a condus la delimitarea a două aspecte fundamentale în utilizarea memoriei, care sunt recunoscute ca **principiul localizării**. Astfel, avem **principiul localizării temporale**: *dacă este referită o entitate atunci este foarte probabil ca aceasta să fie referită din nou, în curând*; şi **principiul localizării spațiale**: *dacă este referită o entitate atunci este foarte probabil ca „vecinii" ei să fie referiți în curând*.

Aplicarea principiului localizării a condus la organizarea ierarhică a memoriei. O **ierarhie de memorie** constă în nivele de memorii cu viteze de acces diferite şi de dimensiuni diferite.

Din punctul de vedere al unui utilizator, memoria se împarte în *memorie externă* şi *memorie internă*. Memoria externă este formată din toate componentele destinate stocării informației, care sunt auxiliare sistemului de calcul. Cele mai folosite memorii externe sunt mediile de memorare pe suport magnetic, adică benzile şi discurile magnetice de toate tipurile. Toate celelalte componente de memorie ale sistemului de calcul formează memoria internă. Datele din memoria externă ajung să fie prelucrate de UCP numai după ce au fost transferate în memoria internă.

Din punctul de vedere al unui programator, memoria trebuie prezentată ierarhic, pe nivele succesive de descriere, aşa cum rezultă din schema următoare:



Săgețile de pe schema de mai sus arată că toate componentele de memorie comunică între ele în ambele sensuri. In plus, memoria cache comunică direct cu UCP.

Când o dată este solicitată de UCP, ea este căutată succesiv în zonele de memorie corespunzătoare nivelelor de mai sus: memorie cache, memorie operativă, memorie secundară, memorie de arhivare. Pentru regăsirea rapidă a datelor căutate, aceste memorii sunt construite astfel încât viteza de acces cea mai mare este la memoria cache şi scade pentru celelalte tipuri, proporțional cu *distanța* față de UCP. In acelaşi timp, creşte capacitatea de memorare, astfel că memoriile de arhivare ajung la capacități suficiente pentru stocarea de baze de date mari şi foarte mari.

In continuare ne vom referi pe rând la tipurile de memorie din ierarhia de mai sus.

Memoria de arhivare sau **memoria auxiliară** este dată de dispozitivele de memorare care se conectează la sistemul de calcul (resurse informaționale). In memoria auxiliară se rețin programele între rulări. Acestea sunt gestionate de utilizator pentru folosirea eficientă a datelor depuse pe ele, de obicei fişiere şi baze de date personale. Aceste suporturi de memorare sunt nevolatile, de obicei discuri magnetice (din 1965).

Ca entitate fizică a SC**,** memoria se mai numeşte **memorie reală.** Ca entitate de utilizare, vom numi memoria **virtuală.** Diferențele dintre acestea depind de modul de organizare fizică şi de complexitatea memoriei în discuție.

Memoria virtuală este întotdeauna considerată ca o succesiune de octeți, în timp ce fizic, poziția a doi octeți vecini în memoria virtuală poate să difere. Legătura dintre un octet considerat în memoria virtuală şi corespondentul său fizic este făcută de mecanismul de adresare.

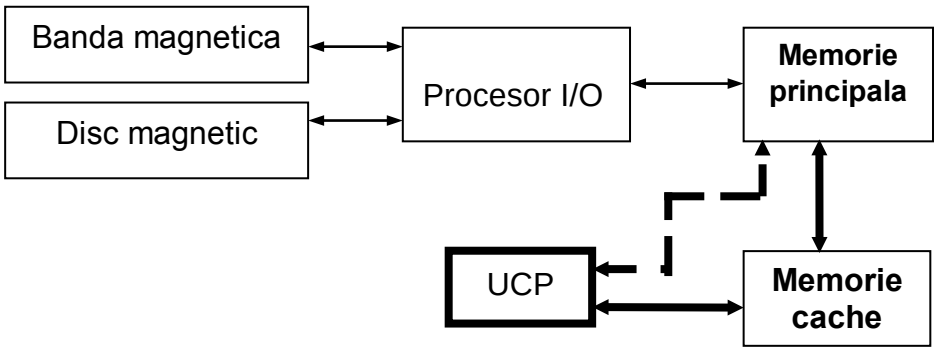
Memoria secundară apare la sistemele care cunosc mecanismul de memorie virtuală. Aceasta reprezintă o **extensie** a memoriei operative. Incepând cu 1985, după ce a apărut sistemul 286 şi modul de lucru protejat, calculatoarele IBM-PC au fost dotate cu **memorie extinsă** care permite o adresare naturală a spațiului de peste un megaoctet de memorie.

Memoria operativă conține datele şi programele pentru toate procesele existente în sistem. O dată cu încheierea unui proces, automat se eliberează zona din memoria operativă alocată procesului respectiv.

Memoria cache (în *engleză, franceză* ascunzătoare, depozit) este memorie internă ultra-rapidă, care comunică direct cu UCP. Este de capacitate mică, dar cu viteză de acces foarte mare. In această zonă sunt reținute în fiecare moment datele cele mai recent utilizate de UCP. Data solicitată de UCP în prelucrarea curentă este adusă din zona de memorie în care a fost găsită şi depusă în memoria cache *împreună cu* un număr de locații vecine datei solicitate, astfel încât, împreună, să umple memoria cache. Această metodă este recunoscută ca **principiul vecinătății** şi este justificat astfel: față de data curent solicitată, este foarte probabil ca următoarea dată solicitată de UCP să fie depusă în memorie într-o locație apropiată de data curentă. O dată ce data curentă a fost adusă în cache împreună cu datele din locații alăturate, rezultă că următoarea regăsire se va face rapid, direct din zona cache. Acest principiu este strâns legat de **principiul localizării**.

In general, vom numi **memorie principală** acea zonă care comunică direct cu UCP. Ea va conține datele şi programele frecvent utilizate de procesor. Memoria principală (primară) este volatilă, implementată cu memorii de tip DRAM (din 1975).

Schema următoare precizează legăturile care se stabilesc între diferite componente ale sistemului de calcul implicate în mecanismul de transfer al datelor între UCP şi memorii.



Din reprezentarea anterioară rezultă că dacă sistemul este recunoscut cu memorie cache atunci aceasta preia funcțiile memoriei principale, fiind direct conectată la UCP şi realizând totodată organizarea transferului de informații între memoria principală şi UCP.

Principalul rol al memoriei cache este de a compensa diferența între viteza de acces la memoria principală şi viteza de prelucrare a procesorului.

De obicei, procesorul este mai rapid decât memoria şi, de aici, activitatea procesorului poate fi încetinită nedorit.

De asemenea, se remarcă prezența procesorului de intrare-ieşire cu rolul de a gestiona transferul datelor între memoria auxiliară şi memoria principală. Se observă că memoria auxiliară nu are acces direct la UCP.

Astăzi există trei tehnologii de bază folosite în construcția ierarhiilor de memorii:

1. memoria principală este implementată cu DRAM-uri;
2. nivelele mai apropiate de UCP (memoriile cache) folosesc SRAM-uri (*Static RAM*);
3. nivelele de memorii lente, dar de capacitate mare folosesc discuri magnetice.

Memoriile de tip SRAM sunt ceva mai scumpe per bit decât memoriile de tip DRAM, deşi sunt considerabil mai lente.

Tehnologia de memorare	Timp de acces standard	$ / MB (1997)
SRAM	5 ns – 25 ns	100$ – 250$
DRAM	60 ns – 120 ns	5$ – 10$
Disc magnetic	10 ms – 20 ms	0,1$ – 0,2$

Intre discurile magnetice şi memoria principală există trei diferențe esențiale:

1. discurile sunt nevolatile – deoarece se bazează pe tehnologii de memorare prin magnetizare (polarizare);
2. discurile au acces lent – deoarece se bazează pe dispozitive mecanice;
3. discurile sunt mai ieftine pe Mega-Byte – deoarece au capacitate de memorare mult mai mare, la un preț rezonabil; în plus, prețul de cost al unei secțiuni de disc este mai mic decât al unui circuit integrat (care stă la baza construcției componentelor memoriei principale, DRAM-uri).

La nivelul anului 1997, 1MB de disc era de 50 de ori mai ieftin decât 1MB de DRAM. Aceasta confirmă faptul că memoriile rapide sunt mai scumpe per bit decât memoriile lente, deci memoriile rapide sunt de obicei de capacitate mai mică.

1.3.3. Memorii cache

La prima maşină comercială care avea un nivel intermediar de memorie între UCP şi memoria principală, acest nivel s-a numit **cache**. Astăzi, deşi acesta este înțelesul propriu al cuvântului cache, termenul este folosit pentru a ne referi şi la ORICE tip de memorie gestionat astfel încât să optimizeze accesul la locația căutată. Deci, **cache** denumeşte o metodă de obținere a unui acces rapid la porțiunile de cod şi de date cele mai recent utilizate.

Memoria cache este organizată în blocuri de lungime fixă, numite linii de cache. O astfel de linie este o copie a unei zone din memoria din care a fost adusă informația în cache.

Proiectarea unei memorii cache trebuie să răspundă la întrebări de tipul:

1. care este dimensiunea optimă a unei linii de cache?
2. cum se regăseşte informația conținută în cache?
3. care linie se înlocuieşte în cazul în care la un transfer memoria cache este plină?

In funcție de modurile de soluționare a acestor probleme, sunt recunoscute mai multe arhitecturi de cache, cum ar fi: memorii cache cu mapare directă, memorii cache asociative, memorii cache set asociative, memorii cache organizate pe sectoare.

1.3.4. Memoria virtuală

După cum memoria cache asigură accesul rapid la informațiile cel mai recent utilizate, *memoria principală poate acționa ca un cache* pentru memoria secundară. Această tehnică se numeşte **memorie virtuală**. Astfel, memoria virtuală este un mecanism folosit pentru a extinde limitele memoriei fizice. Într-un sistem cu memorie virtuală, aceasta apare utilizatorului ca şi cum integral spațiul logic de adrese este disponibil pentru memorare. De fapt, în orice moment, numai câteva pagini din spațiul logic de adrese sunt mapate peste spațiul fizic. Alte pagini (nici) nu sunt prezente (nici) în memoria principală. În schimb, informația din aceste pagini este memorată temporar într-o memorie secundară de tip disc (al cărui cost pe bit este mult mai scăzut). Ori de câte ori este accesată o pagină care lipseşte, sistemul de operare încarcă pagina respectivă de pe disc şi memorează pe disc o altă pagină, care nu a fost recent referită. Prin acest mecanism, utilizatorul are impresia unei memorii fizice uriaşe, dar mai lente.

Există două funcțiuni majore pentru memoria virtuală:

1. să permită partajarea eficientă şi sigură a memoriei între mai multe programe;
2. să permită unui program utilizator unic să mărească logic dimensiunile memoriei primare (principale).

Pentru justificarea primei motivații considerăm un set de programe care se execută simultan pe o maşină. Memoria totală solicitată de toate aceste programe poate fi mult mai mare decât cantitatea de memorie principală disponibilă pe maşină, dar, numai o parte din această memorie este EFECTIV folosită la un moment dat. Memoria principală are nevoie numai de *secvențele active* (procesele curente) ale programelor principale — la fel cum în cache se găseşte la un moment dat numai secvența activă (procesul curent) ale UNUI program. Evident, pentru a permite mai multor programe să partajeze aceeaşi memorie, trebuie ca programele să fi fost protejate unele de altele (un program poate să scrie sau să citească NUMAI zona de memorie care i-a fost asigurată).

De exemplu, la compilarea simultană a mai multor programe nu se poate şti care programe partajează memoria în fiecare moment. De fapt, programele care partajează se modifică dinamic (în timpul execuției). Datorită acestei interacțiuni dinamice trebuie să se compileze fiecare program în propriul lui *spațiu de adresă*. Acesta va fi o secvență de zone de memorie separată, accesibilă numai programului respectiv.

Mecanismul de adresare asociat memoriei virtuale asigură translatarea spațiului de adresă al unui program în adrese fizice. Acest proces de traducere impune protecția spațiilor de adrese între ele.

Pentru justificarea celei de-a doua motivații a utilității memoriei virtuale considerăm următoarea situație. Formal, dacă un program a devenit prea mare pentru memorie atunci este anunțat programatorul. Pentru a determina cauza necesarului mare de memorie, programatorul împarte programul în secvențe şi identifică porțiunile de cod care se exclud reciproc. Aceste depăşiri (*overlays*) sunt sau nu încărcate, sub controlul programului utilizator. Depăşirile de acest tip se organizează în module, fiecare conținând atât date cât şi cod. Apelurile între procedurile aflate în module diferite pot sugera programatorului încapsularea modulelor.

Toate aceste responsabilități sunt preluate din grija programatorului de către memoria virtuală, care va gestiona automat nivelele de memorie principală (DRAM-uri) şi memorie secundară (hard-discuri) din structura ierarhică de organizare a memoriei.

In contextul memoriei virtuale, memoria principală se mai numeşte şi **memorie fizică**. O adresă din memoria principală este o **adresă fizică**. Mulțimea acestor adrese se va numi **spațiu de adrese**. In contextul memoriei virtuale, UCP lucrează cu **adrese virtuale**, care sunt translatate (printr-un mecanism combinat hard şi soft) în adrese fizice.

O **adresă logică** este o adresă utilizată de un programator într-o instrucțiune. Adresele care pot fi utilizate de un program constituie spațiul de adrese logice. Organizarea acestui spațiu de adrese defineşte arhitectura memoriei.

In plus, **memoria virtuală** simplifică tehnica de încărcare a programelor pentru execuție prin introducerea relocării. **Relocarea** aranjează adresele virtuale folosite de un program la diferite adrese fizice, INAINTE ca adresele să fie folosite pentru a accesa memoria. Acest mecanism de relocare permite încărcarea programului în orice locație a memoriei principale cu acelaşi consum de resurse.

Există două tehnici independente de implementare a memoriei virtuale: segmentarea şi paginarea. Cele două tehnici pot fi combinate în scopul extinderii virtuale a spațiului de adrese şi pentru un control mai eficient al alocării spațiului de memorare. Ambele metode presupun divizarea spațiului de memorie (internă şi externă) în blocuri. La execuția unui program se încarcă în memoria internă operativă numai acele blocuri care sunt necesare pentru execuție.

Segmentarea presupune divizarea memoriei în zone compacte numite segmente. Acestea pot să aibă lungime fixă sau variabilă. Ele pot să ocupe intervale disjuncte în spațiul de adrese sau pot să se suprapună total sau

parțial. Adresarea unei locații de memorie se face prin precizarea segmentului şi a adresei relative în cadrul segmentului respectiv. Un segment este caracterizat de: adresa de început a sa, lungime, drepturi de acces. Incărcarea segmentelor în memorie şi amplasarea acestora în spațiul de adrese disponibil sunt sarcini ale sistemului de operare. Locul fizic în care se află o anumită variabilă sau secvență de program este în general transparent pentru programator. Implementarea efectivă a segmentării depinde de suportul hardware oferit de procesor. De exemplu, la modelele moderne de procesoare Intel (începând cu 80386), segmentele logice sunt definite prin descriptori.

Paginarea presupune divizarea memoriei în zone de lungime fixă numite pagini. Paginile sunt încărcate în memoria internă operativă numai dacă sunt solicitate informații conținute în acestea. Paginarea este gestionată de sistemul de operare şi este transparentă pentru utilizator.

1.3.5. Memorii cu acces aleator – RAM şi ROM

In sistemele de calcul sunt utilizate frecvent două structuri constructive de memorie: **RAM** (random access memory) şi **ROM** (read only memory).

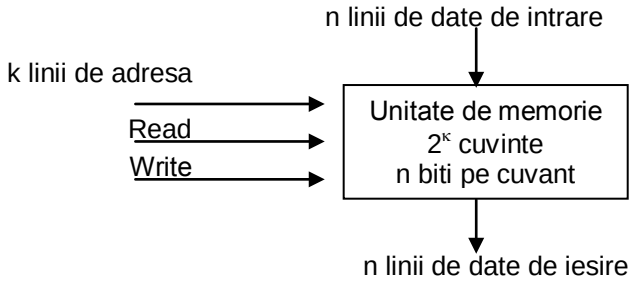
Memoria **RAM** este memorie internă în care se efectuează prelucrările. In RAM se memorează cea mai mare parte a programelor, precum şi datele care se modifică în timpul funcționării sistemului de calcul.

Acest tip de memorii sunt cu *acces direct*, deci informațiile din RAM pot fi accesate pentru transfer din orice zonă, aleasă aleator. De aceea, procesul de localizare a unei date în RAM este acelaşi pentru toate locațiile şi cere un timp mediu egal, indiferent de poziția fizică în memorie a locației respective. De aici provine numele de *acces aleator*.

Memoria RAM este volatilă, deci conținutul ei se pierde la deconectarea electrică a sistemului.

Din 1975, au reținut atenția memoriile DRAM (*Dynamic RAM*). Mai multe memorii DRAM sunt folosite împreună pentru a reține datele şi instrucțiunile programului în execuție.

Un model de bloc de memorie RAM este prezentat în figura următoare.



Comunicarea cu memoria RAM se face prin linii de date de intrare-ieşire, linii de adresă şi linii de control care precizează direcția de transfer a datelor.

Cele k linii de adresă furnizează un număr binar pe k biți care precizează care cuvânt să fie ales din cei 2^k existenți în memorie. Cele două intrări de control (*Read* şi *Write*) dau direcția de transfer a datelor solicitate.

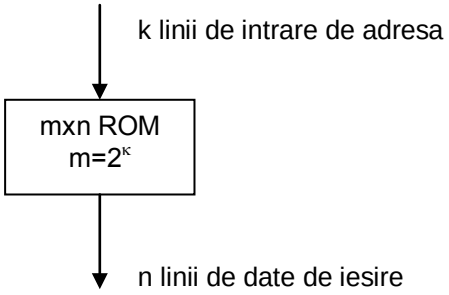
Memoria **ROM** este de dimensiuni mici şi permite numai citirea informațiilor din ea. Aceste informații sunt scrise în memoria **ROM** printr-o tehnologie specială de către firma producătoare a calculatorului o dată pentru totdeauna.

In sistemul de calcul, un ROM este folosit pentru memorarea programelor fixe (care nu se modifică în timpul funcționării SC) şi a tabelelor de constante care nu-şi schimbă valoarea în sistem.

Intr-o unitate de control, o memorie ROM poate fi folosită pentru conservarea informațiilor codificate care reprezintă secvențe de variabile interne de control necesare pentru activarea a diferite operații în sistem. O unitate de control care foloseşte ROM pentru memorarea informațiilor binare de control este numită *unitate de control cu microprogram* (*microprogrammed control unit*).

Memoria ROM este nevolatilă, deci conținutul ei este permanent, adică nu este influențat de deconectarea sistemului de la rețeaua electrică.

O memorie ROM de tip *mxn* este un tablou de celule binare organizate în *m cuvinte* a câte *n biți* fiecare. Pentru $m=2^k$ avem o memorie ROM de tipul:



Memoria ROM nu are nevoie de linii de control de tip *Read* deoarece în fiecare moment cele *n* linii de ieşire furnizează automat cei *n* biți ai cuvântului selectat prin valoarea adresei de intrare. Mai mult, deoarece valorile biților în ROM au valori fixe, acest tip de memorie nu are nevoie de circuite proprii de stocare a datelor, aşa cum întâlnim la RAM.

Observație. Inițial, acronimul RAM era folosit pentru memorie cu acces aleator la informația stocată. Astăzi, el este folosit pentru memorii de tip *Read/Write* pentru a îl diferenția de memoria ROM, care este şi ea cu acces aleator la informație.

1.4. Proiectarea calculatoarelor moderne

După Patterson şi Hennessy [5], se pot anunța patru principii de bază ale proiectării tehnologiei hardware, pe care le enumerăm în continuare.

Principiul 1. Simplitatea favorizează uniformitatea [5, pag.97].

Principiul 2. Mai mic înseamnă mai rapid [5, pag.99].

Principiul 3. Proiectarea bună necesită compromisuri bune [5, pag.108].

Principiul 4. Cazul frecvent trebuie făcut să aibă execuție rapidă [5, pag.133].

Rezumat.

Totalitatea componentelor destinate stocării datelor formează unitatea de memorie a sistemului de calcul. Acest paragraf detaliază aspecte concrete legate de structura şi organizarea memoriei calculatoarelor moderne. Astfel, sunt abordate diferite puncte de vedere: (1) structura fizică a memoriei, (2) structura ierarhică de organizare, (3) structura internă dependentă de tipologia memoriei.

Cuvinte cheie.

bit, bit de informație, bistabil
locație de memorie, octet, cuvânt
principiul localizării, principiul vecinătății, cache
memorie cu acces aleator

Verificare.

1. Calculați capacitatea totală de memorare (în megabytes) a unei dischete cu două fețe şi care este formatată cu următoarele constante: 80 piste (cilindri), 18 sectoare, 524 octeți pe bloc (un bloc constă într-un sector de pe o pistă). Capacitatea unui bloc este dată de 512 octeți de date propriu-zise şi 12 octeți auxiliari (necesari marcării începutului blocului şi localizării acestuia).
2. Uzual, o pagină de text are 40 linii cu 75 de caractere pe linie. Câte pagini de astfel de text se pot memora pe un CD de 600MB?
3. Folosiți principiile de organizare a memoriei cache pentru a justifica caracteristicile acesteia.
4. Care este capacitatea unei memorii cu 12 linii de intrare de adresă şi 16 linii de ieşire (intrare) pentru date?
5. Incercați să justificați cele patru principii de bază ale proiectării hardware enunțate de Patterson şi Hennessy.

NOTIȚE

CAPITOLUL 2

Componentele digitale ale unui sistem de calcul

CUPRINS

CAPITOLUL 2 ---------- **49**

2.1. CIRCUITE LOGICE ---------- **51**
2.1.1. Porţi logice. CLC. CLS ---------- *51*
2.1.2. CLC aritmetice ---------- *54*
2.1.2.1. CLC sumator ---------- 54
2.1.2.2. CLC scăzător ---------- 58
2.1.2.3. CLC sumator cu comutare ---------- 61
2.1.3. CLC decodoare ---------- *63*
2.1.4. CLC multiplexoare ---------- *64*
2.1.5. CLS bistabile ---------- *66*
2.1.6. Circuite integrate ---------- *71*
2.2. REGIŞTRI ---------- **74**
2.2.1. Regiştri cu încărcare paralelă ---------- *74*
2.2.2. Regiştri de deplasare ---------- *75*
2.2.3. Numărătoare binare ---------- *77*
2.2.4. Regiştri de incrementare ---------- *79*
2.3. STRUCTURA DE BAZĂ A UNUI CALCULATOR ---------- **81**
2.3.1. Regiştrii principali ai unui sistem de calcul ---------- *81*
2.3.2. Unitatea de contorizare şi control ---------- *82*
2.3.3. Transferul datelor pe magistrală ---------- *82*

Un sistem digital este un ansamblu de componente care concură la prelucrarea datelor şi la executarea instrucțiunilor primite. Aceste componente constau din circuite logice, grupate în componente aritmetice, decodoare, regiştrii ş.a. Conectate la magistralele sistemului, aceste componente interacționează şi astfel, îndeplinesc misiunea primită de la utilizator.

In capitolul de față vom decrie aceste componente interne.

2.1. Circuite logice

2.1.1. Porți logice. CLC. CLS

Informația binară este reprezentată în sistemul de calcul digital prin cantități fizice, numite **semnale**, care se transmit conform cu construcția internă a fiecărei componente. Cele mai simple semnale electrice sunt de tip on / off.

Pentru descrierea funcționării calculatorului este util să construim pentru componentele de bază traseul semnalelor electrice, adică să prezentăm componentele respective prin **circuitele logice** interne.

Semnalele electrice sunt date de anumite tensiuni şi se regăsesc în sistem în una din două stări posibile. Aceste stări corespund valorilor unei variabile binare, adică 0 sau 1. De exemplu, un SC particular poate lucra cu o tensiune de 3V pentru starea corespunzătoare valorii 1 şi o tensiune de 0.5V pentru starea corespunzătoare valorii 0. In acest caz, intrările în CLC recunosc tensiuni de 0.5V şi 3V, iar la ieşire circuitul *răspunde* cu o tensiune egală tot cu una dintre aceste valori.

Pentru buna înțelegere a celor ce urmează este util să introducem aici câteva noțiuni.

Dacă notăm $\mathbf{B_2} = \{0,1\}$ şi $\mathbf{B_2^n} = \mathbf{B_2 x B_2 x ... x B_2}$ atunci sunt imediate următoarele definiții:

Definiția 1. Se numeşte **funcție logică** o aplicație $f : \mathbf{B_2^n} \longrightarrow \mathbf{B_2^m}$.

Definiția 2. Dacă m = 1 atunci **funcția logică** se numeşte **scalară**. Altfel, spunem că avem o **funcție logică vectorială**.

Reprezentarea unei funcții logice se poate da prin tabelă de valori (tabelă de adevăr), prin expresie logică sau prin circuitul logic asociat.

In practică se remarcă unele funcții, numite **funcții logice elementare**: **ŞI LOGIC** (conjuncția), **SAU LOGIC** (disjuncția), **NU LOGIC** (negarea sau complementarea). Circuitele logice asociate acestor funcții se numesc **porți logice elementare**.

Convenim să notăm conjuncția multiplicativ, disjuncția aditiv şi negarea prin operatorul *apostrof*.

Caracterizarea funcțiilor logice elementare este dată în tabelul următor:

Nume	*Simbol grafic*	*Expresie logică*
ŞI LOGIC	a, b, x	x = a b sau x = a b
SAU LOGIC	a, b, x	x = a + b
NU LOGIC	a, x	x = a’

Tabelele de valori pentru aceste funcții sunt:

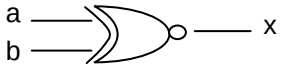
a	b	a b
0	0	0
0	1	0
1	0	0
1	1	1

a	b	a+b
0	0	0
0	1	1
1	0	1
1	1	1

a	a’
0	1
1	0

Pe lângă porțile logice elementare, reprezentarea cu circuite apelează la porțile logice ale unor funcții uzuale. Acestea sunt: **NAND**, **NOR**, **XOR** şi **NXOR**, definite şi caracterizarte conform cu următorul tabel:

Nume	*Simbol grafic*	*Expresie logică*
NAND **şi – nu**	a, b, x	x = (a b)’
NOR **sau – nu**	a, b, x	x = (a + b)’
XOR **sau exclusiv** **(anticoincidență)**	a, b, x	x = a’b + ab’ = = a ⊕ b

NXOR **NOR exclusiv** **(echivalență)** **(coincidență)**		x = ab + a'b' = = (a ⊕ b)'

Funcțiile uzuale introduse anterior sunt definite de următoarele tabele de valori:

a	b	(a b)'
0	0	1
0	1	1
1	0	1
1	1	0

a	b	(a+b)'
0	0	1
0	1	0
1	0	0
1	1	0

a	b	a ⊕ b
0	0	0
0	1	1
1	0	1
1	1	0

a	b	(a⊕ b)'
0	0	1
0	1	0
1	0	0
1	1	1

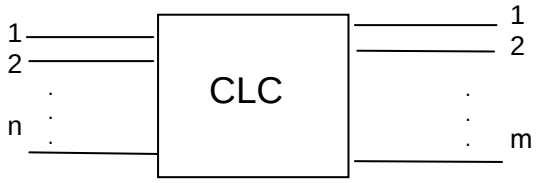
Pentru simplificarea funcțiilor logice trebuiesc cunoscute un minim de identități din algebra booleană:

(1)	a + 0 = a	(10)	a 0 = 0
(2)	a + 1 = 1	(11)	a 1 = a
(3)	a + a = a	(12)	a a = a
(4)	a + a' = 1	(13)	a a' = 0
(5)	a + b = b + a	(14)	ab = ba
(6)	a + (b + c) = (a + b) + c	(15)	a(bc) = (ab)c
(7)	a(b + c) = ab + ac	(16)	a + bc = (a + b)(a + c)
(8)	(a + b)' = a'b'	(17)	(ab)' = a' + b'
(9)	(a')' = a		

Definiția 3. Dacă o funcție este reprezentată prin circuit logic atunci argumentele funcției sunt **intrări în circuit**, iar valorile funcției, pentru fiecare combinație a valorilor de intrare, reprezintă **ieşirile din circuit**.

Definiția 4. Un **circuit logic combinațional** (**CLC**) este un circuit logic în care valorile de pe ieşiri depind numai de valorile de intrare în circuit. Altfel, dacă ieşirile depind şi de intrări şi de stările intermediare ale componentelor interne ale circuitului, atunci avem un **circuit logic secvențial** (**CLS**).

Prin convenție, un circuit logic combinațional cu n intrări şi m ieşiri se reprezintă astfel:



In paragrafele următoare vom detalia analiza unor circuite logice care se remarcă prin importanța pe care o au în sistemele de calcul.

2.1.2. CLC aritmetice

Pentru efectuarea operațiilor aritmetice sistemul de calcul este prevăzut cu componente specializate care alcătuiesc unitatea aritmetico-logică, UAL. Structura internă a acestor componente este dată de circuite logice combinaționale construite astfel încât să determine la ieşire rezultatul operării valorilor de intrare.

In acest paragraf ne referim la componentele UAL pentru efectuarea operațiilor aritmetice.

2.1.2.1. CLC sumator

Fie două numere naturale A şi B date în reprezentare binară pe n poziții prin:

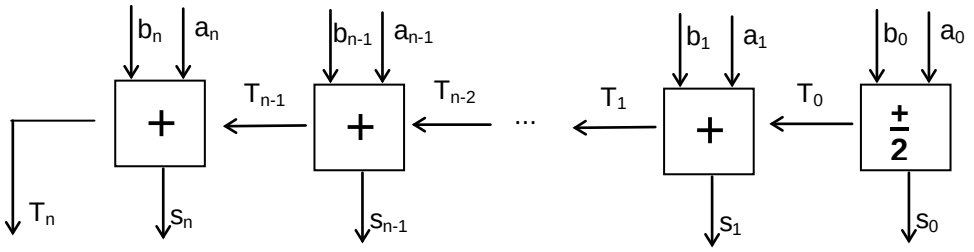
$$A = a_n a_{n-1} \ldots a_2 a_1 a_0 \text{ şi}$$

$$B = b_n b_{n-1} \ldots b_2 b_1 b_0.$$

Observație. Dacă cifrele semnificative ale unuia dintre numere nu ocupă toate cele n poziții atunci acestea se încarcă prin aliniere la dreapta şi zona neocupată se încarcă cu zerouri.

Notăm cu S suma celor două numere A şi B. Reprezentarea binară a lui S este $S = s_n s_{n-1} \ldots s_2 s_1 s_0$.

Scopul acestui paragraf este să construiască un circuit logic combinațional care să aibă ca intrări valorile cifrelor binare ale operanzilor şi ca ieşiri cifrele binare ale sumei, numit **CLC sumator total**. Schematic, acesta se reprezintă astfel:



Am notat cu T_i, i = 0, 1,…n-1,n cifra de transport de la rangul i la rangul i + 1. Deoarece cifra de transport de la ultimul rang, T_n, face parte din rezultat, ieşirea aceasta s-a legat la masă. In consecință, ea va fi citită ca şi cifră rezultat împreună cu cifrele s_0, s_1, … , s_n ale sumei S.

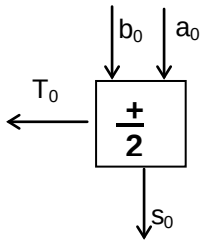
In reprezentarea de mai sus se disting două tipuri de circuite, şi anume:

a) circuit de adunare cu două intrări şi două ieşiri, numit **circuit semisumator** şi

b) circuit de adunare cu trei intrări şi două ieşiri, numit **circuit sumator complet**.

Avem circuit semisumator pentru operare pe rangul zero, în care se adună cifrele operanzilor şi rezultă cifra sumei şi cifra de transport. In rest, pentru operarea pe celelalte ranguri, avem circuite sumatoare complete, în care se adună cifrele operanzilor cu cifra de transport de la rangul anterior şi rezultă cifra sumei şi cifra de transport la rangul următor.

a) Circuitul semisumator se reprezintă schematic astfel:



Fiecare ieşire corespunde unei funcții logice. Pentru descrierea funcționării circuitului vom determina mai întâi expresiile logice ale acestor funcții.

Tabla adunării în baza 2 determină următorul tabel de valori pentru funcțiile s_0 şi T_0:

a	b	a+b
0	0	0
0	1	1
1	0	1
1	1	10

=>

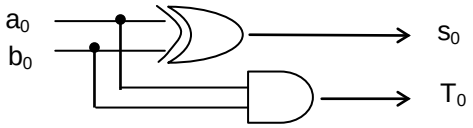
a_0	b_0	s_0	T_0
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

De aici putem deduce următoarele expresii logice:

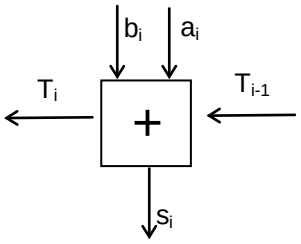
$s_0 = a_0'b_0 + a_0b_0' = a_0 \oplus b_0$ şi

$T_0 = a_0b_0$.

Corespunzător, circuitul logic semisumator se reprezintă prin următoarele porți logice:



b) Circuitul sumator complet pentru adunarea cifrelor binare de rang i se reprezintă schematic astfel:



Pentru descrierea funcționării circuitului vom proceda ca şi în cazul circuitului semisumator. Pentru început, determinăm expresiile logice ale funcțiilor s_i şi T_i de ieşire.

Avem următorul tabel de valori pentru funcțiile s_i şi T_i:

a_i	b_i	T_{i-1}	s_i	T_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

De aici putem deduce următoarele expresii logice:

$$s_i = a_i'b_i'T_{i-1} + a_i'b_iT'_{i-1} + a_ib_i'T'_{i-1} + a_ib_iT_{i-1} =$$
$$= (a_i'b_i' + a_ib_i)\, T_{i-1} + (a_i'b_i + a_ib_i')\, T'_{i-1} =$$
$$= (a_i \oplus b_i)'T_{i-1} + (a_i \oplus b_i)\, T'_{i-1} =$$
$$= T_{i-1} \oplus (a_i \oplus b_i)$$

şi

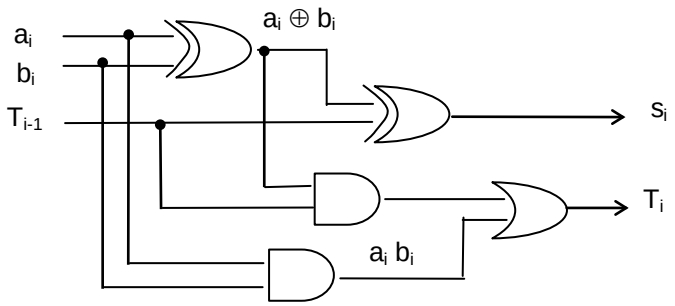
$T_i = a_i'b_iT_{i-1} + a_ib_i'T_{i-1} + a_ib_iT'_{i-1} + a_ib_iT_{i-1} =$

$= (a_i'b_i + a_ib_i')\ T_{i-1} + a_ib_i\ (T'_{i-1} + T_{i-1}) =$

$= (a_i \oplus b_i)\ T_{i-1} + a_ib_i =$

$= a_ib_i + T_{i-1}(a_i \oplus b_i).$

Corespunzător, circuitul logic sumator complet se reprezintă prin următoarele porți logice:



Pentru a reprezenta circuitul sumator total mai avem un singur pas, care constă în determinarea unei expresii pentru T_i care să nu fie recursivă.

Notăm

$P_i := a_i \oplus b_i$, $(\forall)$ i= 0,1,2,…, n şi

$G_i := a_ib_i$, $(\forall)$ i= 0,1,2,…, n.

Cu aceste notații, sunt imediate următoarele relații:

$s_0 = P_0$, $T_0 = G_0$,

şi

$T_1 = G_1 + P_1T_0 = G_1 + P_1G_0$,

$T_2 = G_2 + P_2T_1 = G_2 + P_2G_1 + P_2P_1G_0$,

$T_3 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0$.

De aici, putem anunța că, pentru orice i, este verificată relația

$T_i = G_i + P_iG_{i-1} + P_iP_{i-1}G_{i-2} + \ldots + P_iP_{i-1} \ldots P_1G_0$.

Evident, relația este complet justificată după aplicarea unei inducții matematice.

Folosind această relație pentru T_i, suntem acum în măsură să construim circuitul sumator total. Pentru simplitatea reprezentării vom alege n=3, adică situația în care fiecare operand are 4 poziții binare.

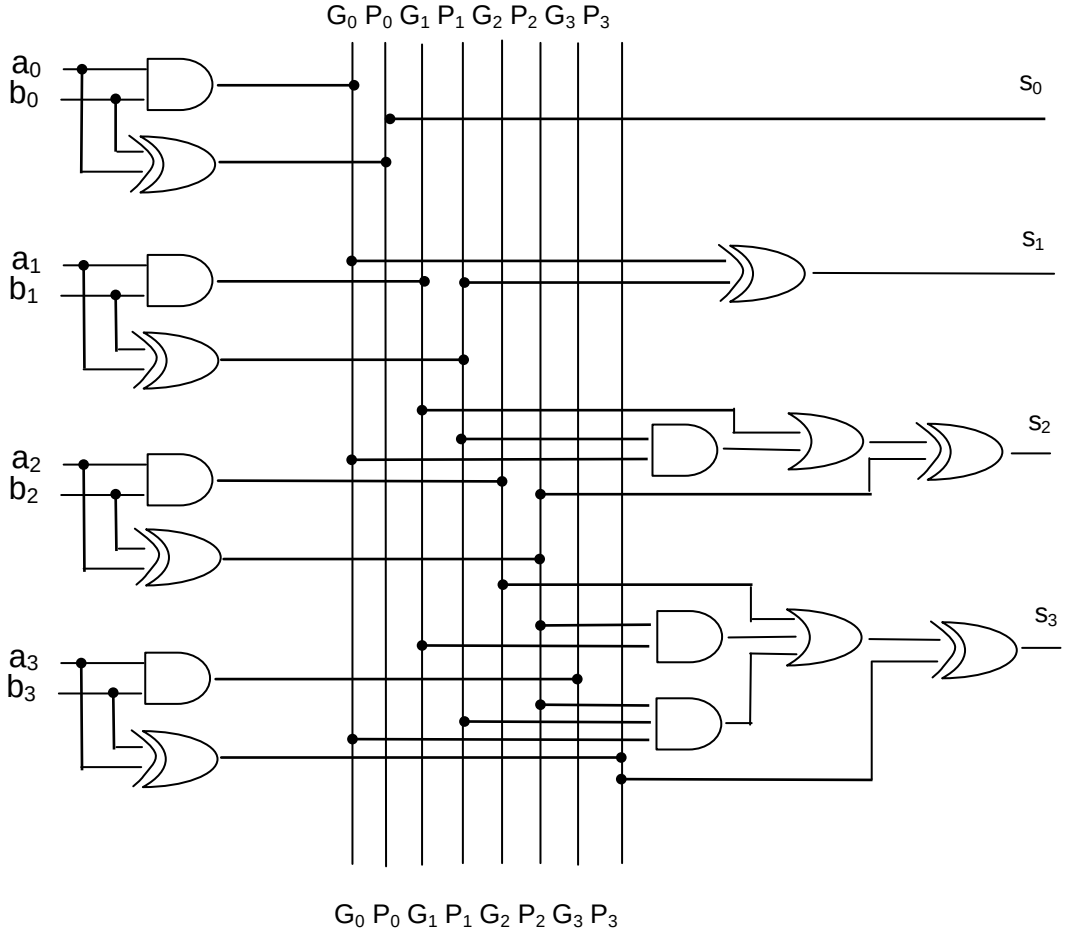
Expresiile logice pentru cifrele sumei sunt:

$s_0 = P_0$;

$s_1 = G_0 \oplus P_1$;

$s_2 = (G_1 + P_1G_0) \oplus P_2$;

$s_3 = (G_2 + P_2G_1 + P_2P_1G_0) \oplus P_3$.



2.1.2.2. CLC scăzător

Urmând paşii construcției din paragraful anterior, vom descrie aici circuitul logic combinațional care determină diferența a două numere naturale.

Fie cele două numere naturale A şi B date în reprezentare binară pe n poziții prin:

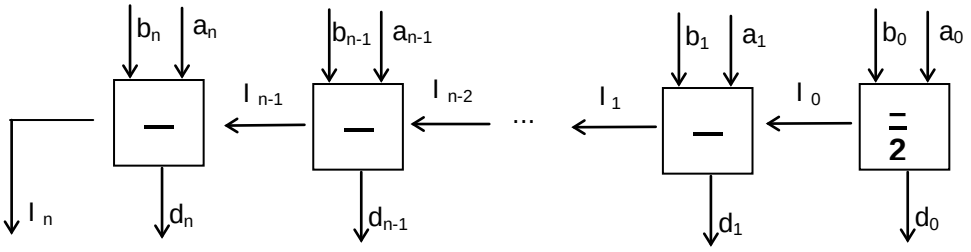
$$A = a_n a_{n-1} \ldots a_2 a_1 a_0 \text{ şi}$$

$$B = b_n b_{n-1} \ldots b_2 b_1 b_0.$$

Notăm cu D diferența celor două numere A şi B. Reprezentarea binară a lui D este $D = d_n d_{n-1} \ldots d_2 d_1 d_0$.

Circuitul logic combinațional scăzător are ca intrări valorile cifrelor binare ale operanzilor şi ca ieşiri cifrele binare ale diferenței.

Schematic, acesta se reprezintă astfel:



Am notat cu I_i, i = 0, 1,…n-1,n cifra de împrumut de la rangul i+1 la rangul i. Deoarece cifra de împrumut pentru ultimul rang, I_n, face parte din rezultat, ieşirea aceasta s-a legat la masă. In consecință, ea va fi citită ca şi cifră rezultat împreună cu cifrele d_0, d_1, … , d_n ale diferenței D.

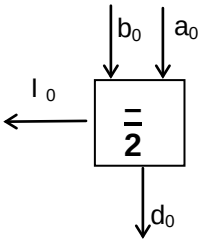
In reprezentarea de mai sus se disting două tipuri de circuite, şi anume:

a) circuit de scădere cu două intrări şi două ieşiri, numit **circuit semiscăzător** şi

b) circuit de scădere cu trei intrări şi două ieşiri, numit **circuit scăzător complet**,

cu precizările introduse la circuitul sumator.

a) Circuitul semiscăzător se reprezintă schematic astfel:



Pentru descrierea funcționării circuitului vom determina expresiile ale funcțiilor logice de ieşire, I_0 şi d_0.

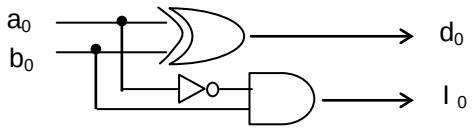
a_0	b_0	d_0	I_0
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

De aici putem deduce următoarele expresii logice:

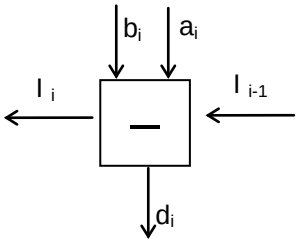
$d_0 = a_0'b_0 + a_0b_0' = a_0 \oplus b_0$ şi

$I_0 = a_0' b_0$.

Corespunzător, circuitul logic semiscăzător se reprezintă prin următoarele porți logice:



b) Circuitul scăzător complet pentru scăderea cifrelor binare de rang i se reprezintă schematic astfel:



Expresiile logice ale funcțiilor de ieşire d_i şi I_i sunt date de următorul tabel de valori. Calculul constă în $a_i - I_{i-1} - b_i$ şi rezultă cifra d_i a diferenței şi împrumutul I_i.

a_i	b_i	I_{i-1}	d_i	I_i
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

De aici putem deduce următoarele expresii logice:

$d_i = a_i'b_i'I_{i-1} + a_i'b_iI'_{i-1} + a_ib_i'I'_{i-1} + a_ib_iI_{i-1} =$

$= (a_i'b_i' + a_ib_i) I_{i-1} + (a_i'b_i + a_ib_i') I'_{i-1} =$

$= (a_i \oplus b_i)'I_{i-1} + (a_i \oplus b_i) I'_{i-1} =$

$= I_{i-1} \oplus (a_i \oplus b_i)$

şi

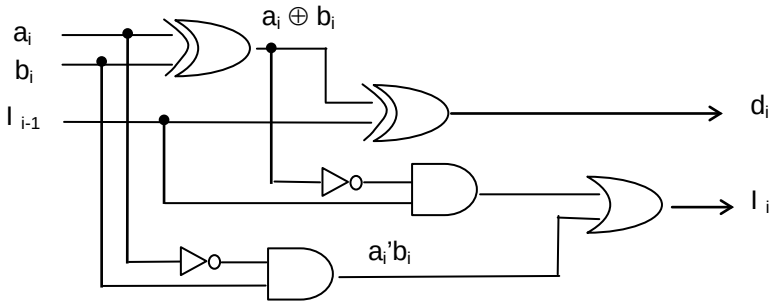
$I_i = a_i'b_i'I_{i-1} + a_i'b_iI'_{i-1} + a_i'b_iI_{i-1} + a_ib_iI_{i-1} =$

$= (a_i'b_i' + a_ib_i)\, I_{i-1} + a_i'b_i\, (I'_{i-1} + I_{i-1}) =$

$= (a_i \oplus b_i)'\, I_{i-1} + a_i'b_i =$

$= a_i'b_i + I_{i-1}(a_i \oplus b_i)'.$

Corespunzător, circuitul logic scăzător complet se reprezintă prin următoarele porți logice:



2.1.2.3. CLC sumator cu comutare

Expresiile logice pentru funcțiile de ieşire din circuitele aritmetice anterioare au fost:

$$s_0 = a_0 \oplus b_0 = d_0$$

$$s_i = T_{i-1} \oplus (a_i \oplus b_i),\ (\forall)\ i= 1..n$$

$$d_i = I_{i-1} \oplus (a_i \oplus b_i),\ (\forall)\ i= 1..n$$

şi, respectiv,

$$T_0 = a_0b_0$$

$$I_0 = a_0'b_0$$

$$T_i= a_ib_i + T_{i-1}(a_i \oplus b_i),\ (\forall)\ i= 1..n$$

$$I_i= a_i'b_i + I_{i-1}(a_i \oplus b_i)',\ (\forall)\ i= 1..n.$$

Analizând aceste formule şi circuitele asociate lor, se observă o mare asemănare şi faptul că celula de scădere are în plus două porți NU care generează termenii a_i' şi $(a_i \oplus b_i)'$.

Pornind de la această observație, se pune problema realizării unui circuit capabil să efectueze atât adunări cât şi scăderi.

Construirea acestui circuit se bazează pe proprietatea de comutare a porții XOR, care constă în următoarea afirmație. Dacă se consideră o poartă XOR cu o intrare de date notată x şi cu a doua intrare notată c atunci funcționarea ei este dată de următoarea tabelă de valori:

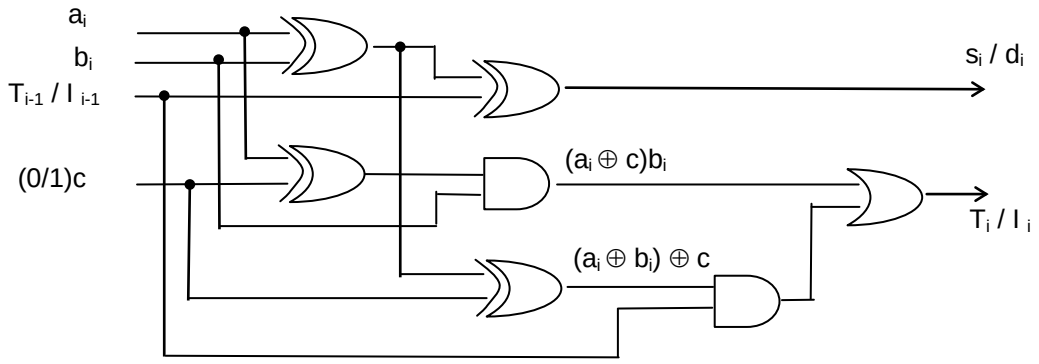
x	c	$y = x \oplus c$
0	0	0
0	1	1
1	0	1
1	1	0

şi poate fi interpretată prin:

$$y = \begin{cases} x, c = 0 \\ x', c = 1 \end{cases}.$$

Astfel, putem denumi intrarea c ca fiind **intrare de comutare** (sau de condiție) deoarece valoarea ei este cea care decide dacă pe ieşirea din poarta XOR avem valoarea de pe intrare sau complementara (negarea) ei.

Cu acestea, se poate construi un circuit care să efectueze atât adunare cât şi scădere, pe care îl vom numi **circuit sumator cu comutare**. Pentru operarea pe rangul i, acest circuit va consta din următoarele porți logice:



Dacă pe intrarea c punem valoarea 0 atunci circuitul funcționează ca sumator, iar pentru c = 1, ca scăzător.

Semnificațiile pentru intrarea T_{i-1} / I_{i-1} şi ieşirile s_i / d_i, T_i / I_i trebuiesc citite la *numărător*, respectiv la *numitor*. Astfel, dacă c=0 atunci valoarea pusă pe intrarea T_{i-1}/I_{i-1} reprezintă cifră de transport (T_{i-1}), iar pentru c=1 reprezintă cifră de împrumut (I_{i-1}). In consecință, valorile pe ieşiri se corespund astfel: s_i şi T_i pentru c=0 şi respectiv d_i şi I_i pentru c=1.

Intrarea de condiție este folosită de două ori şi anume pentru comutarea valorii lui a_i şi a valorii $a_i \oplus b_i$. Prin comutarea valorii lui a_i se asigură că, pentru c=0, la ieşirea din poarta XOR avem a_i, iar pentru c=1, la ieşirea din poarta XOR avem a_i'. In acest fel, formula pentru valoarea depusă pe ieşirea a doua foloseşte pe a_i, dacă c=0 şi pe a_i', dacă c=1. Acest mod de calcul este corect şi respectă formulele logice pentru T_i ($T_i = a_ib_i + \ldots$), respectiv I_i ($I_i = a_i'b_i + \ldots$).

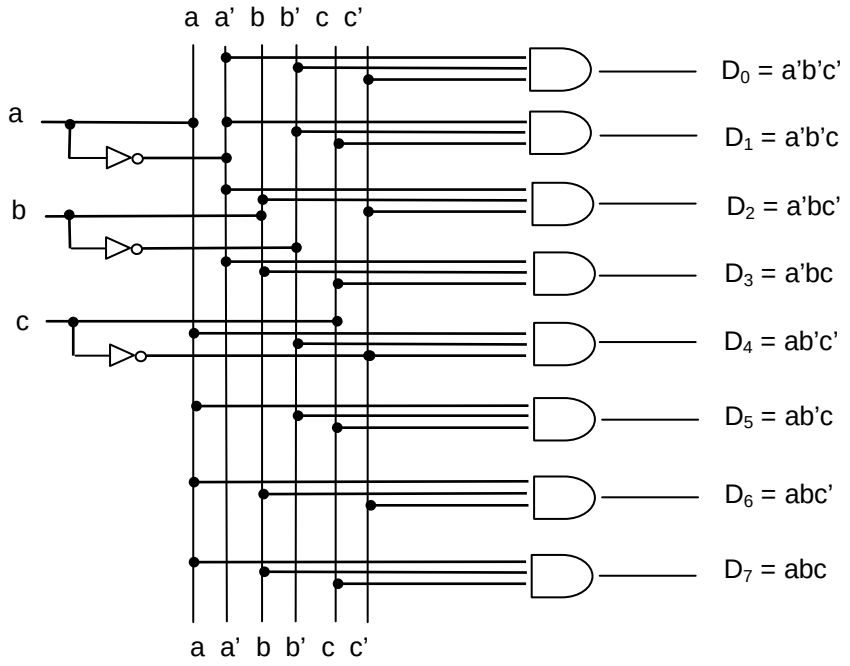
2.1.3. CLC decodoare

In sistemul de calcul informația este reprezentată prin codificare binară. Un *cod binar de n biți* poate să reprezinte până la 2^n informații codificate distincte.

Un **decodor** este un circuit logic combinațional care converteşte informația dată binar pe n intrări în cele 2^n ieşiri posibile. Dacă, dintre cele n intrări, unele sunt invalidate atunci decodorul va avea mai puțin de 2^n ieşiri.

Să considerăm un decodor cu trei intrări, a, b, c, toate valide. Rezultă că acesta va avea $8 = 2^3$ ieşiri. Algebric, funcțiile pentru cele 8 ieşiri sunt date de expresiile celor 8 termeni canonici de tip produs: a'b'c', a'b'c, a'bc', a'bc, ab'c', ab'c, abc', abc.

Un astfel de decodor se mai numeşte de tip 3x8.

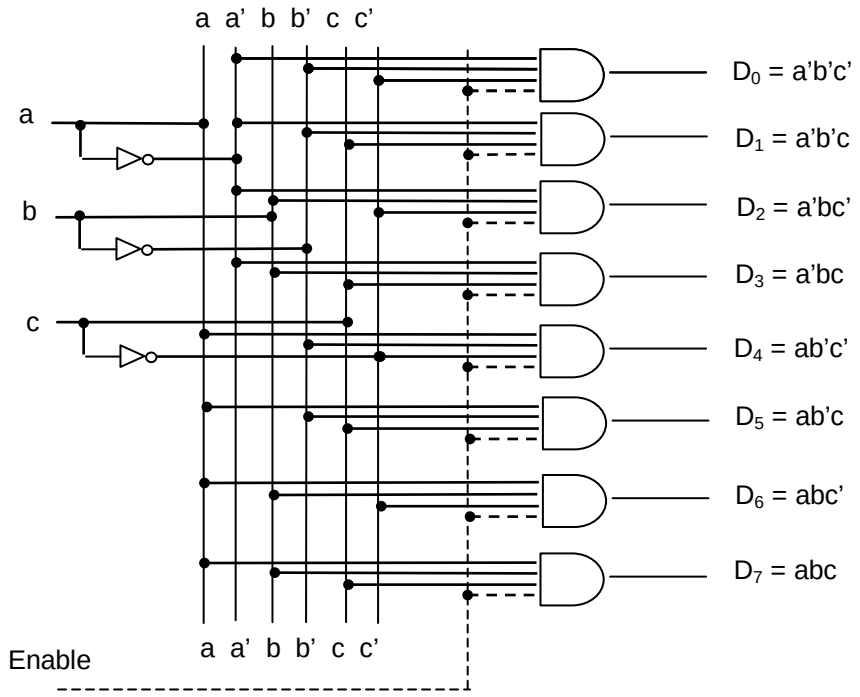


Deoarece termenii canonici implementați pe ieşiri sunt disjunctivi, rezultă că, indiferent de combinația valorilor pe intrări, o singură ieşire va avea valoare 1 logic, celelalte având valoarea 0 logic.

O aplicație imediată a acestui decodor este conversia numerelor din binar în octal. Concret, reprezentarea binară dată pe intrarea în decodor corespunde cifrei octale egală cu rangul ieşirii care are valoarea 1 logic.

abc	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
000	1	0	0	0	0	0	0	0
001	0	1	0	0	0	0	0	0
010	0	0	1	0	0	0	0	0
011	0	0	0	1	0	0	0	0
100	0	0	0	0	1	0	0	0
101	0	0	0	0	0	1	0	0
110	0	0	0	0	0	0	1	0
111	0	0	0	0	0	0	0	1

Pentru controlul funcționării unui decodor, de multe ori acesta este prevăzut cu o **intrare de validare** (*Enable*). Dacă pe această intrare avem 0 logic atunci circuitul nu funcționează adică, toate ieşirile sunt pe 0. Dacă pe intrarea *Enable* avem 1 logic atunci circuitul funcționează ca un decodor fără validare.

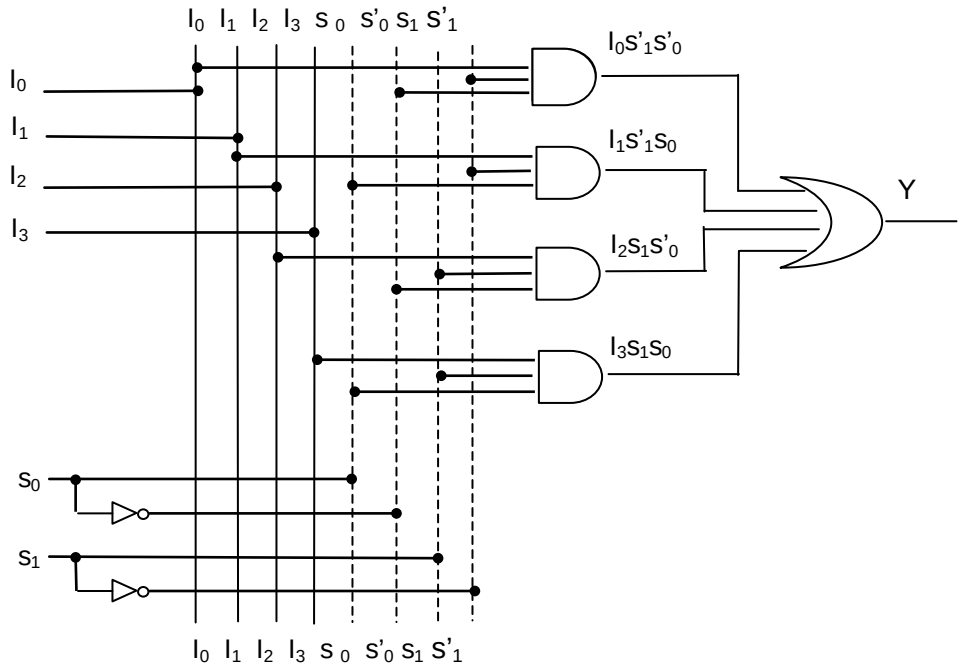


2.1.4. CLC multiplexoare

Un **multiplexor** este un circuit logic combinațional care primeşte informație binară de pe una din cele 2^n linii de date de intrare şi o direcționează spre ieşirea sa unică. Linia de date este selectată prin n linii de selecție (linii de adresă).

Un multiplexor de tip 2^nx1 are 2^n linii de date de intrare şi n linii de selecție, ale căror combinații de biți determină care dintre intrările de pe liniile de date este selectată pentru ieşirea Y.

In figura următoare considerăm un multiplexor de tip 4x1.



Acest multiplexor are 6 intrări = 4 de date + 2 de selecție şi o ieşire. Selecția intrării se face conform cu următoarea tabelă:

S_1	S_0	Y	
0	0	I_0	$S'_1S'_0$
0	1	I_1	S'_1S_0
1	0	I_2	$S_1S'_0$
1	1	I_3	S_1S_0

Pe coloana a patra am trecut monoamele elementare, care definesc selectarea ieşirii aşa cum rezultă din reprezentarea circuitului.

Observații. Datorită funcției de a selecta una dintre intrările în circuit, un multiplexor se mai numeşte şi **selector de date** (*data selector*). Abrevierea pentru un multiplexor este **MUX**.

2.1.5. CLS bistabile

Cele mai multe dintre sistemele de calcul, pe lângă circuitele combinaționale (CLC) au nevoie (în special pentru componentele de memorare) şi de circuite logice secvențiale (CLS). Dintre acestea, cele mai folosite sunt CLS sincrone sau cu sincronizare în timp. Un **CLS sincron** lucrează cu semnale care modifică conținutul zonelor de memorie numai la momente de timp discrete, bine determinate. Sincronizarea este asigurată de un generator de impulsuri de tact (de ceas).

Un **circuit basculant bistabil** (pe scurt CBB sau bistabil – dacă se ține cont de timpul de propagare a semnalului electric prin liniile de circuit, atunci termenii nu sunt sinonimi [Tanenbaum, pag. 134]) este o structură de memorare bazată pe CLS sincrone. Un bistabil îşi menține starea binară curentă până când primeşte un impuls de ceas care *intră* în ansamblul de porți logice şi determină schimbarea de stare. Trecerea dintr-o stare în alta se face numai la primirea unui impuls de tact extern.

Fiecare bistabil este capabil să memoreze un bit de informație.

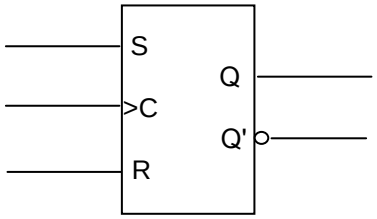
Bistabilii se diferențiază prin numărul de intrări şi prin felul în care valorile de intrare influențează starea bistabilului. Cele mai folosite tipuri de bistabili sunt:

- CBB de tip SR;
- CBB de tip D;
- CBB de tip JK;
- CBB de tip T.

Pentru început vom prezenta aceste circuite în cazul asincron, urmând ca, în final, să prezentăm şi modelul sincron pentru bistabilul de bază, SR.

CBB de tip SR are trei intrări notate respectiv cu: S pentru **S**etare, R pentru **R**esetare şi C pentru intrarea de ceas şi o ieşire, notată Q (quit). Auxiliar, un astfel de bistabil poate avea o ieşire suplimentară cu valoare egală cu ieşirea principală negată.

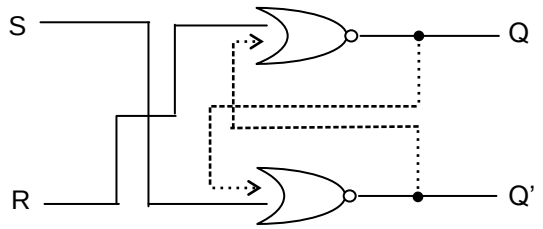
Simbolic, acest bistabil se reprezintă astfel:



Intrarea de ceas este marcată cu caracterul „>”. Un ceas este un circuit care emite o serie de pulsuri cu o durată de puls precisă şi cu un interval precis între pulsuri succesive. Intervalul de timp între fronturile

corespunzătoare a două pulsuri succesive se numeşte **durată de ciclu** (engl., *clock cycle time*). Frecvențele de puls (numărul de pulsuri în unitatea de timp) sunt în general între 1 şi 500 MHz, corespunzând perioadei de ceas de la 1000 ns la 2 ns.

Porțile logice care asigură funcționarea acestui bistabil sunt două porți NOR interconectate. Prin *interconectare* înțelegem că ieşirile sunt repuse pe intrări prin *bucle de reacție*. In figura următoare buclele de reacție sunt figurate cu linie punctată.



Notăm cu Q_t valoarea pe ieşirea Q a bistabilului la momentul t şi cu Q_{t+1} valoarea pe aceeaşi ieşire la momentul următor, t+1. Cu alte cuvinte, Q_t dă starea bistabilului înainte de aplicarea semnalelor S şi R pe intrare, iar Q_{t+1} dă starea bistabilului după aplicarea semnalelor respective.

Datorită existenței buclelor de reacție, rezultă că starea Q_{t+1} depinde de intrările S şi R, dar şi de starea anterioară, Q_t.

Conform legării porților logice în reprezentarea de mai sus, expresia logică pentru Q_{t+1} este

$$Q_{t+1} = (R + (S + Q_t)')' = R' (S + Q_t) = R'S + R'Q_t.$$

Această expresie determină următoarea tabelă de valori pentru schimbarea de stare (Q_t –> Q_{t+1}) a bistabilului de tip SR:

S	R	Q_t	Q_{t+1}	
0	0	0	0	fără modificare de stare
0	0	1	1	(Q_{t+1} coincide cu Q_t)
0	1	0	0	punere pe 0 (resetare)
0	1	1	0	
1	0	0	1	punere pe 1 (setare)
1	0	1	1	
1	1	0	0(*)	stare nedeterminată
1	1	1	0(*)	(comenzi interzise)

Observație. Pentru intrarea (1,1), funcționarea teoretică a circuitului determină valoarea 0 pentru Q_{t+1}, indiferent de starea anterioară a bistabilului,

Q_t. Cu toate acestea, practic, datorită întârzierilor care apar în sistem la transmiterea semnalelor, starea Q_{t+1} este nedeterminată (nu se stabilizează, chiar după mai multe reveniri pe buclele de reacție). In acest caz spunem că *funcționarea bistabilului este ambiguă*.

Din tabelul anterior rezultă cum depinde starea următoare Q_{t+1} de valorile puse pe intrările în bistabil, S şi R.

S	**R**	Q_{t+1}	
0	0	Q_t	fără modificare de stare
0	1	0	resetare = punere pe 0
1	0	1	setare = punere pe 1
1	1	?	situație imprecisă (nedeterminată)

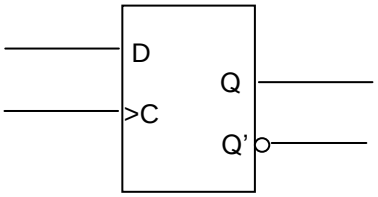
Din afirmațiile anterioare se poate trage concluzia că un bistabil de tip SR nu funcționează întotdeauna corect. De aceea, se impune stabilirea unei relații pe care trebuie să o verifice valorile puse pe intrările S şi R astfel încât bistabilul să nu intre într-o stare nedeterminată. Această ecuație este

$$SR = 0,$$

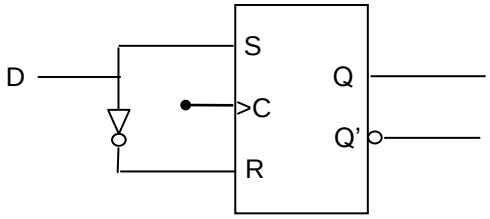
şi ea asigură că cel puțin una dintre valorile de intrare este 0. Cu alte cuvinte, se evită tocmai situația în care ambele intrări sunt pe 1 logic.

CBB de tip D (**D**ata) este uşor modificat față de CBB – SR. Acesta are o singură intrare de date (D), intrarea de tact (C) şi ieşirile identice cu tipul SR. Pe intrarea D se inserează un inversor, astfel că se obțin două direcții care vor corespunde intrărilor S şi R ale tipului SR.

Simbolul este:



Un bistabil de tip D se obține din SR prin următoarea modificare:



Din această modalitate de a lega intrarea D la intrările S şi R rezultă:

✓ a avea intrarea D în 0 logic este echivalent cu a avea 0 pe intrarea S şi 1 pe intrarea R;

✓ a avea intrarea D în 1 logic este echivalent cu a avea 1 pe intrarea S şi 0 pe intrarea R.

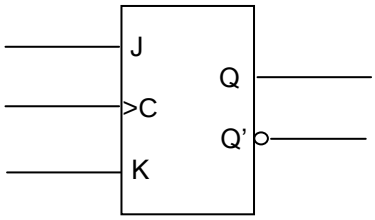
Se observă că oricare ar fi valoarea pusă pe intrarea în bistabilul de tip D, acesta funcționează corect, deoarece este întotdeauna verificată relația

$$SR = 0.$$

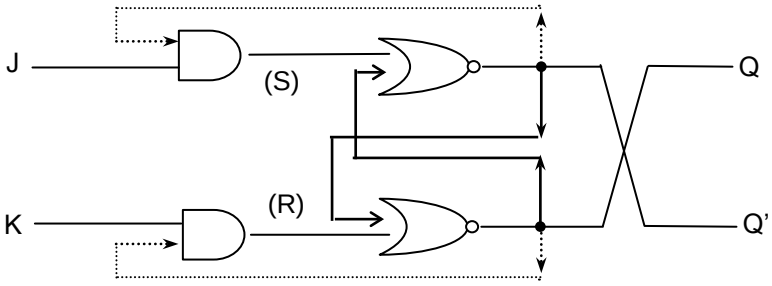
Funcționarea bistabilului de tip SR determină următoarea tabelă de funcționare pentru bistabilul de tip D:

D	Q_{t+1}	
0	0	resetare = punere pe 0
1	1	setare = punere pe 1

CBB de tip JK are două intrări de date, J şi K, intrarea de tact, C şi ieşirile identice cu tipul SR.



Din punctul de vedere al circuitelor interne, un bistabil JK este unul SR completat cu două porți SI şi două *linii de reacție totală*. In figura următoare liniile de reacție totală sunt marcate cu linie punctată.



Această structură internă determină schimbarea de stare Q_t –> Q_{t+1} conform cu următorul tabel:

J	K	Q_t	Q_{t+1}	
0	0	0	0	fără modificare de stare
0	0	1	1	(Q_{t+1} coincide cu Q_t)
0	1	0	0	punere pe 0 (resetare)
0	1	1	0	

1	0	0	1	punere pe 1 (setare)
1	0	1	1	
1	1	0	1	trecere în starea
1	1	1	0	complementară

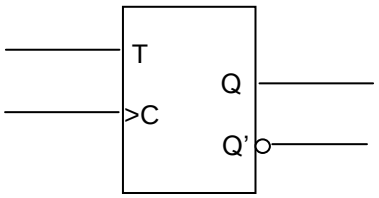
Corespunzător, tabela de funcționare pentru bistabilul de tip JK este următoarea:

J	**K**	Q_{t+1}	
0	0	Q_t	Fără modificare de stare
0	1	0	Resetare = punere pe 0
1	0	1	Setare = punere pe 1
1	1	Q'_t	Trecere în starea complementară

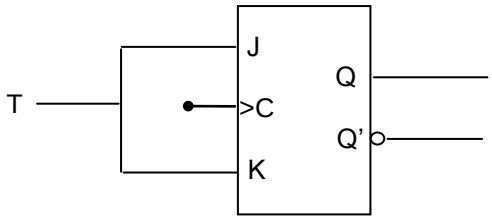
Observație. Despre bistabilul de tip JK putem spune că este primul model de circuit secvențial veritabil. Aceasta deoarece, când avem (1,1) pe intrările J şi K, starea următoare Q_{t+1} se poate determina numai dacă se cunoaşte starea anterioară Q_t.

CBB de tip T (**T**oggle = piron, cârjă) este un model de bistabil care funcționează în doi timpi. Această afirmație va fi pe deplin înțeleasă după prezentarea funcționării circuitului.

Simbolul de reprezentare este:



Un bistabil de tip T se obține din unul JK prin legarea intrărilor J şi K la aceeaşi intrare T, astfel:

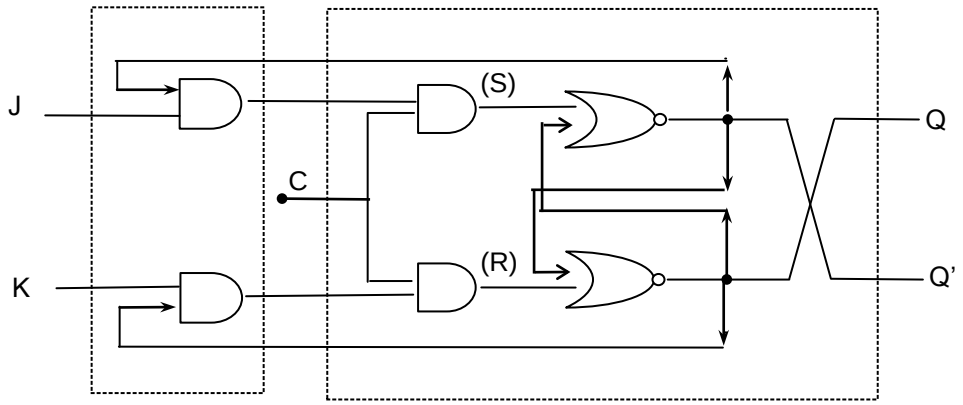


Din această construcție rezultă că bistabilul de tip T are două variante de funcționare, care coincid cu tipul JK pentru J = K = 0 şi J = K = 1:

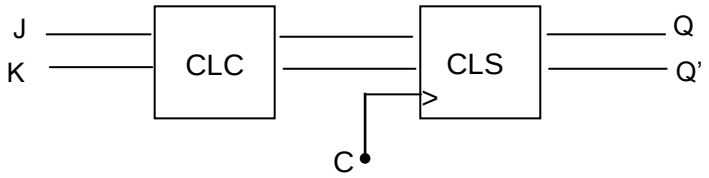
T	Q_{t+1}	
0	Q_t	Fără modificare de stare
1	Q'_t	Trecere în starea complementară

Observație. Cât timp pe intrarea T avem 1 logic, circuitul intră în starea complementară a stării curente şi, apoi, la următoarea schimbare de stare, revine în starea inițială. De aceea, se spune despre acest bistabil că funcționează în doi timpi.

In încheierea paragrafului referitor la bistabili prezentăm modelul sincron pentru bistabilul de tip JK. Se va vedea că acesta completează modelul asincron cu două porți SI care preiau valoarea logică pusă pe intrarea de tact, C.



In această reprezentare am pus în evidență şi secțiunile asincronă şi sincronă ale circuitului, care corespund celor două componente de bază: un CLC şi un CLS.



2.1.6. Circuite integrate

Componentele digitale ale unui sistem de calcul sunt realizate cu circuite logice integrate. Un **circuit integrat** (CI) este format dintr-un semiconductor de siliciu, de dimensiuni foarte reduse, numit **chip**, care conține componentele porților logice. Mai multe porți logice sunt interconectate in interiorul chip-ului astfel încât să formeze circuitul cerut. Chip-ul este închis într-o carcasă de ceramică sau de material plastic, iar conexiunile sunt sudate cu conductoare fine de aur la pinii de ieşire pentru a

forma circuitul integrat. Numărul pinilor poate varia de la 14 pini la un pachet integrat mic, până la 1000 sau mai mulți la un circuit mare.

O dată cu îmbunătățirea tehnologiilor de realizare a circuitelor integrate, a crescut considerabil şi numărul porților care pot fi implementate pe un acelaşi chip. După acest criteriu, diferența între chip-urile care au câteva porți şi cele care au sute de porți componente se face prin numele acestora. Astfel, avem următoarele clase: CI pe scară joasă, CI pe scară medie, CI pe scară largă şi CI pe scară foarte largă. Scara de integrare se referă la numărul de componente electronice pe unitatea de suprafață a circuitului integrat.

Dăm în tabelul următor câteva caracteristici pentru aceste clase:

SSI	Small Scale Integration	mai puțin de 10 porți	IN-OUT sunt conectate direct la pinii circuitului
MSI	Medium Scale Integration	10-200 porți	folosite în decodoare, sumatoare, regiştri
LSI	Large Scale Integration	200-câteva sute de porți	includ procesoare, chipuri de memorie, module programabile
VLSI	Very Large Scale Integration	mii de porți	dimensiuni mici, preț redus, deci foarte folosite în sistemele moderne.

Circuitele logice integrate se pot clasifica şi după tehnologia de fabricare, rezultând *familii de circuite logice*. Fiecare familie logică de circuite este reprezentată de un tip de circuit pe baza căruia se dezvoltă toate circuitele din acea familie. Comercial vorbind, structurile care s-au impus sunt:

- CI de tip TTL (*Transistor – transistor logic*);
- CI de tip ECL (*Emitter – coupled logic*);
- CI de tip MOS (*Metal – oxide semiconductor*);

Caracterizarea acestor tipuri de circuite integrate poate fi făcută astfel:

TTL	originea = DTL (diode – transistor logic) care implementa poarta NAND cu diode şi tranzistori; ulterior s-au înlocuit diodele cu tranzistori şi, de aceea repetăm cuvântul trazistor în denumire	tensiunea maximă aplicabilă este de 5V, iar valorile logice 0 şi 1 corespund pentru 0V şi respectiv 3.5V;

ECL	folosite în sisteme de viteză foarte mare (supercalculatoare, procesoare de semnale)	tranzistorii operează într-o stare nesaturată, condiție care determină o întârziere în propagarea semnalului de maxim 1-2 nanosecunde;
MOS	este un tip de CI care foloseşte tranzistori unipolari, față de TTL şi ECL care folosesc tranzistori bipolari.	

Rezumat.

Acest paragraf are ca obiectiv principal definirea principalelor circuite logice care intră în structura unui circuit elementar de memorie (bistabil), a unui registru (vezi paragraful al doilea) şi/sau în structura unei UAL elementare. Tipurile de circuite sunt prezentate gradat, de la simplu la complex, aşa cum se poate vedea şi din lista de cuvinte cheie.

Pentru toate aceste prezentări teoretice de modele, în culegerea de probleme sunt date exemple concrete de utilizare şi interconectare a circuitelor. Structura culegerii de probleme urmăreşte paragraf cu paragraf cursul, astfel încât studenții pot rezolva problemele pas cu pas, pe măsură ce sunt parcurse la curs elementele teoretice. Ultimul paragraf al culegerii de probleme conține o serie de exemple de sinteză, unele fiind chiar modele concrete de circuite interne ale calculatorului.

Cuvinte cheie.

circuit logic
poartă logică, poartă logică elementară, poartă logică uzuală
CLC, CLC aritmetic (sumator, scăzător, sumator cu comutare), decodor, multiplexor
CLS, CLS bistabil (SR, D, JK, T)
circuit integrat

NOTIȚE

2.2. Regiştri

Un **registru** este un ansamblu de bistabili (circuite basculante bistabile, CBB) şi, eventual, porți logice care realizează schimbările de stare ale bistabililor. Grupul de bistabili care formează registrul pot fi încărcați simultan, sub acțiunea unui impuls de tact unic.

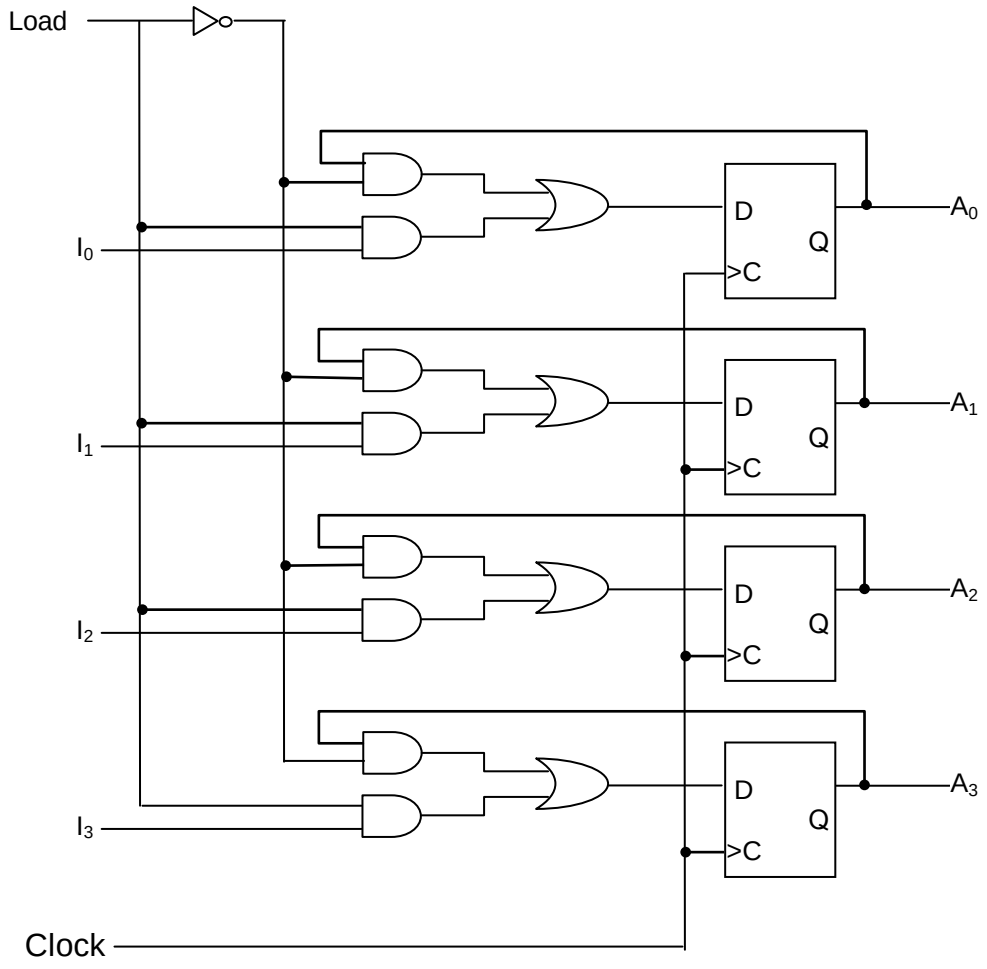
Un **registru de *n* biți** are *n* bistabili şi poate memora orice informație reprezentabilă pe *n* biți. Porțile logice au rolul de a controla când şi cum o informație nouă este transferată în registru.

Componentele unui sistem digital sunt complet definite de regiştrii pe care îi conțin şi de operațiile care se execută asupra datelor lor. Operațiile executate asupra datelor memorate în regiştrii se numesc **microoperații**. Exemple: deplasare, numărare, încărcare, ştergere.

Pentru executarea fiecăreia dintre aceste microoperații există structuri bine determinate ale regiştrilor respectivi. In paragrafele următoare vom descrie structurile unora dintre aceşti regiştri.

2.2.1. Regiştri cu încărcare paralelă

Structura unui astfel de registru este următoarea:



Funcția unui registru cu încărcare paralelă este de a prelua valorile logice puse pe intrările de date, simultan pentru toate pozițiile binare şi la primirea aceluiaşi impuls de ceas.

Acest registru constă din patru bistabili de tip D, legați prin porți logice de cele patru intrări de date I_0, I_1, I_2, I_3 şi de intrarea de validare, Load.

Rolul intrării Load este de a dirija preluarea datelor de pe intrările I_0, I_1, I_2, I_3 numai la primirea unui impuls de tact. Dacă pe intrarea Load avem 0 logic atunci intrările de date sunt inhibate şi intrările D în bistabili sunt conectate respectiv cu ieşirile Q. Dacă pe Load avem 1 logic atunci valorile depuse pe intrările I_0, I_1, I_2, I_3, sunt încărcate în registru.

Valorile logice considerate a fi memorate în registru în fiecare moment sunt cele de pe liniile A_0, A_1, A_2, A_3.

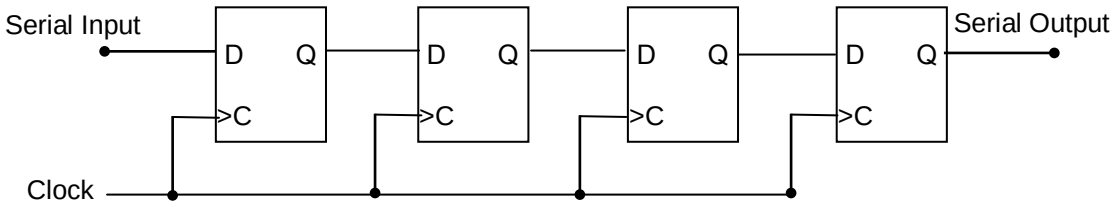
Bistabilul de tip D nu are variantă de funcționare *fără modificare de stare* ci, la fiecare impuls de tact primit, valoarea preluată de bistabil pe intrarea D este cea care determină următoarea stare la ieşire. Pentru a păstra valoarea la ieşire nemodificată, cât timp nu se primeşte impuls de tact, este necesar să menținem intrarea D egală cu valoarea curentă de pe ieşire. Această funcționare este asigurată de conexiunile de reacție, care leagă fiecare ieşire Q la intrarea corespunzătoare în poartă logică SI şi, mai departe la intrarea D a aceluiaşi bistabil.

Impulsul de tact se aplică pe intrarea Clock în fiecare moment. Cu toate acestea, dacă circuitul este dotat cu o intrare de validare de tip Load, atunci ea este cea care decide dacă următorul impuls de ceas va determina încărcarea simultană a datelor depuse pe intrările de date sau preluarea pe liniile de reacție a valorilor curente memorate în registru. Cu alte cuvinte, intrarea Load decide dacă registrul primeşte informație nouă sau informația memorată rămâne nemodificată.

2.2.2. Regiştri de deplasare

Un registru capabil să deplaseze, adică să translateze cu o poziție informația memorată de bistabilii săi se numşte **registru de deplasare**. In funcție de construcția registrului, operația de mutare (*shift*) se poate face spre stânga, spre dreapta sau în ambele direcții.

Reprezentarea este:

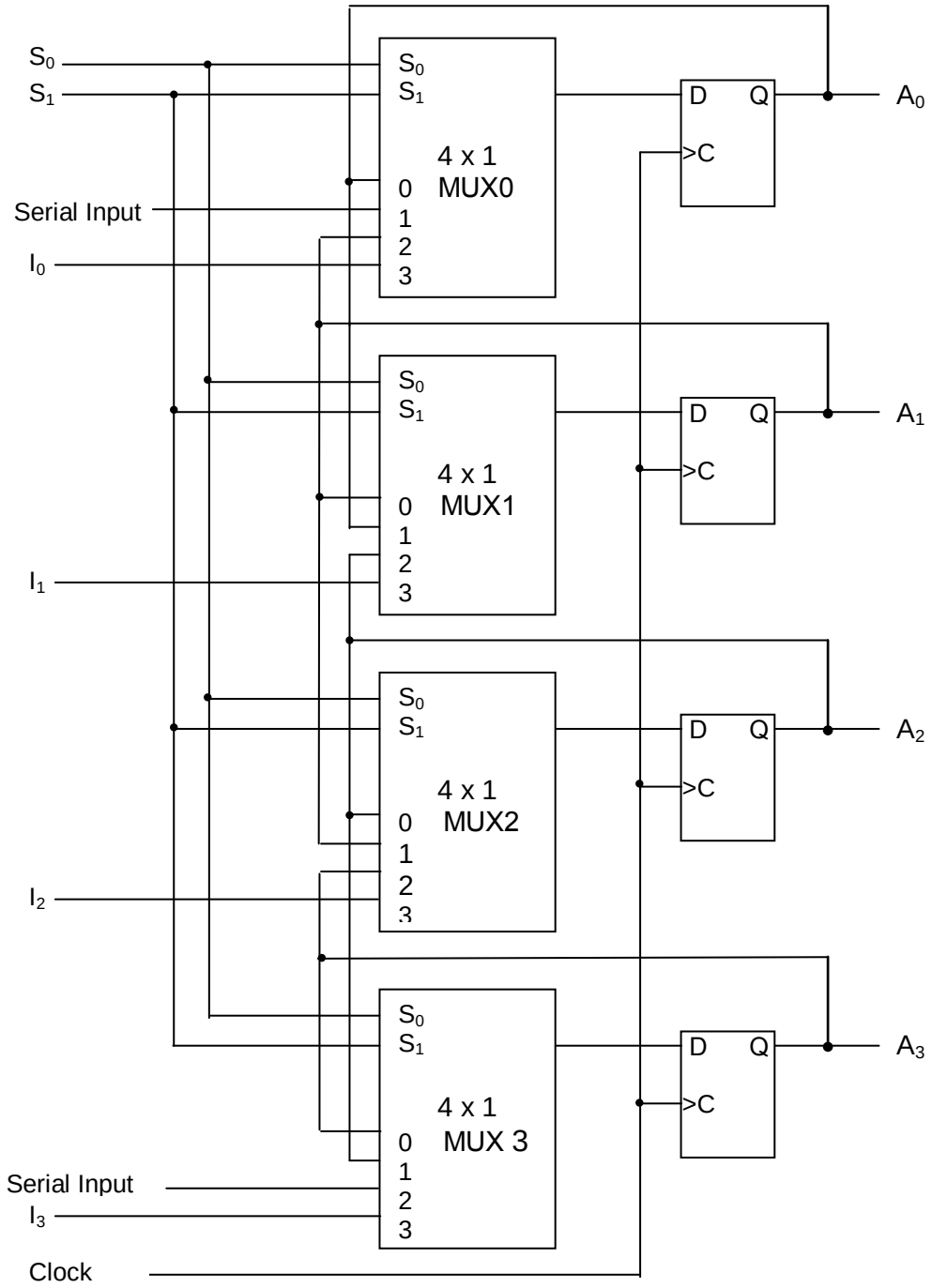


Se observă că ieşirea dintr-un bistabil este intrare în următorul. In plus, funcționarea este sincronizată prin faptul că toate intrările de ceas sunt legate

la aceeaşi sursă. Atfel, toţi bistabilii vor primi la acelaşi moment impulsul de tact care determină deplasarea informaţiei.

Intrarea *Serial Input* conţine informaţia care se va memora în primul bistabil, iar ieşirea *Serial Output* va conţine vechiul conţinut al ultimului bistabil al registrului de deplasare analizat.

Detaliem acest tip de registru prezentând structura internă a unui registru pe 4 biţi care poate executa sau o deplasare, sau o încărcare paralelă (simultană), în funcţie de valorile puse pe intrările de selecţie ale circuitului. Registrul constă din patru multiplexoare conectate la patru bistabili de tip D.



Reamintim că valorile memorate în registru la un moment dat sunt cele de pe liniile A_0, A_1, A_2, A_3 şi ele se modifică numai simultan, la primirea unui nou impuls de tact.

Din schemă se observă că:

- fiecare MUX preia pe intrarea sa 0 valoarea curentă din registru, transmisă pe linia de reacție;
- fiecare MUX preia pe intrarea sa 1 valoarea din locația de rang cu 1 mai mic;
- fiecare MUX preia pe intrarea sa 2 valoarea din locația de rang cu 1 mai mare;
- fiecare MUX preia pe intrarea sa 3 valoarea depusă pe intrarea de date corespunzătoare.

Această organizare este în concordanță cu următoarea tabelă de funcționare. In funcție de valorile depuse pe intrările de selecție S_0 şi S_1, avem:

S_1	S_0	Intrarea selectată de fiecare MUX	
0	0	0	Fără modificare de stare
0	1	1	Shift Right (down)
1	0	2	Shift Left (up)
1	1	3	Incărcare paralelă

Cele două intrări *Serial Input* introduc valori noi în registru pe pozițiile eliberate de operația de deplasare.

2.2.3. Numărătoare binare

O numărătoare este un registru care, la aplicarea impulsurilor de ceas, urmează o succesiune prefixată de stări. In particular, numărătoarele binare vor lucra pe secvențe de valori binare. Aplicațiile imediate ale unui registru numărătoare sunt:

- contorizarea numărului de apariții ale unui eveniment;
- cronometrarea necesară generării semnalelor de control pentru secvențele de operații de executat în sistem.

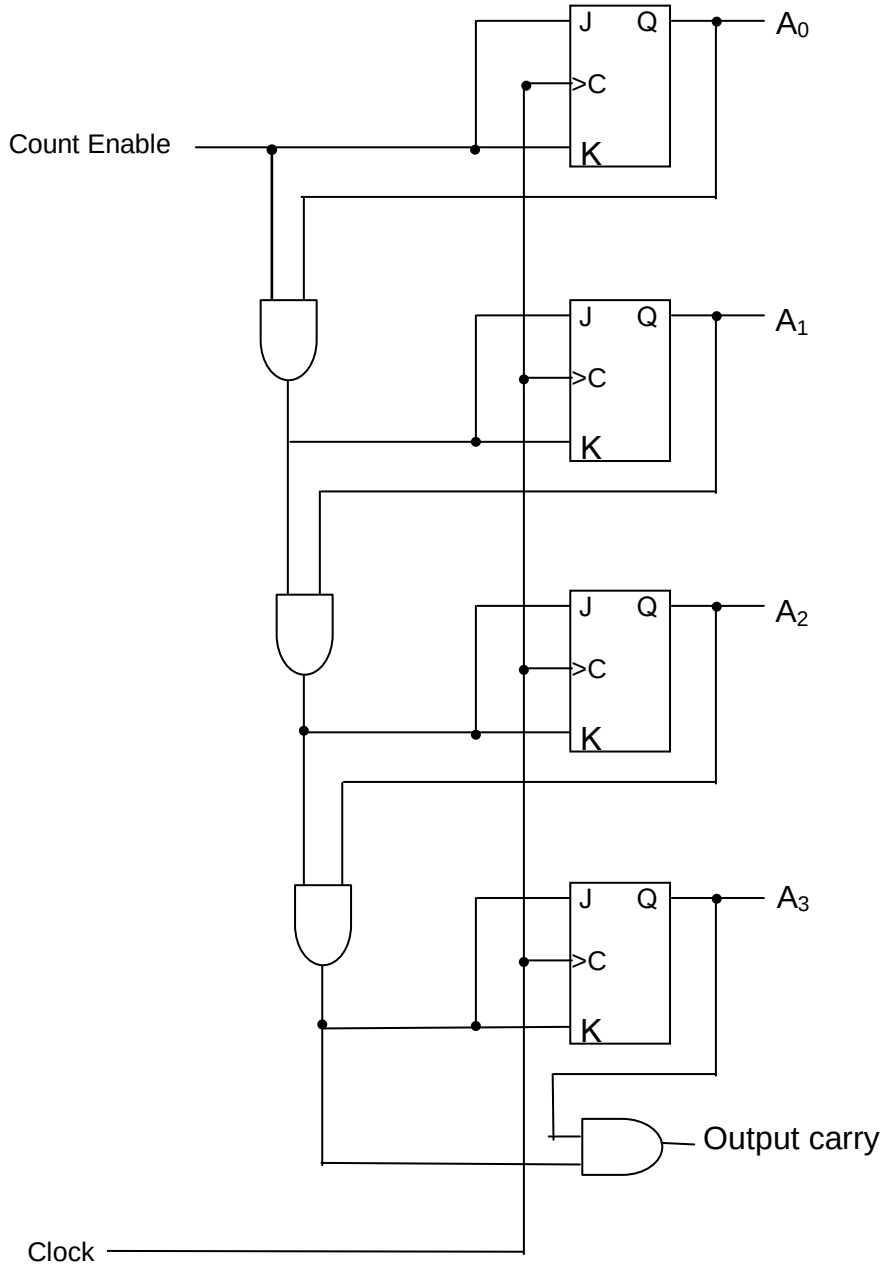
Pentru o numărătoare binară pe 4 biți succesiunea de stări prin care trece este următoarea: 0000 –> 0001 –> 0010 –> 0011 –> 0100 –> 0101 –> 0110 –> 0111 –> 1000 –> 1001 –> 1010 –> 1011 –> 1100 –> 1101 –> 1110 –> 1111.

Realizarea circuitului de numărare constă în transcrierea cu porți logice a următoarei observații: pentru a trece dintr-o stare în alta trebuie efectuate operațiile:

1. LSB (bitul cel mai puțin semnificativ) se complementează;
2. fiecare dintre ceilalți biți se complementează numai dacă în configurația anterioară a avut toți biții de rang mai mic egali cu 1.

De aici rezultă că acest registru trebuie să folosească bistabili care au posibilitatea de a trece în starea complementară a stării curente, deci bistabili de tip JK sau T.

Concret, un registru de numărare pe patru biți are următoarea structură:



Intrarea *Count Enable* este una de validare a funcționării registrului, deci decide care sunt impulsurile de tact care determină trecerea

numărătoarei din starea curentă în starea următoare. Aşa cum este construit circuitul, schimbarea de stare are loc când pe *Count Enable* avem 1 logic.

Conform cu operația a doua care trebuie efectuată la orice schimbare de stare, valoarea curentă din fiecare locație a registrului influențează valoarea încărcată în fiecare dintre pozițiile de rang mai mare. Această proprietate este asigurată prin faptul că fiecare linie A_i este legată „în cascadă” prin porți SI la intrările în toți bistabilii de rang mai mare.

Ieşirea *Output Carry* are semnificație de cifră de transport. Aceasta este folosită în situația în care configurația curentă este 1111 şi numărătoarea ar trebui să treacă în configurația următoare, 10000. Având numai patru biți, registrul va reține 0000, iar bitul 1 va fi transmis pe ieşirea de transport, *Output Carry*.

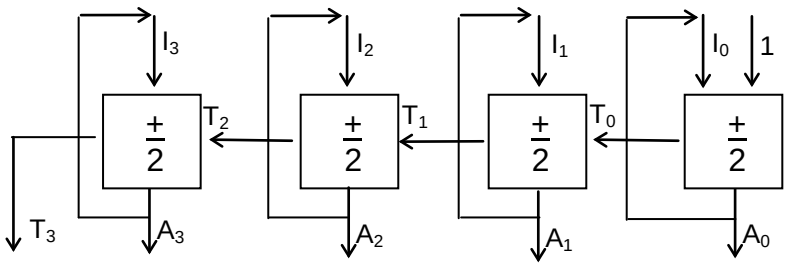
2.2.4. Regiştri de incrementare

Un registru de incrementare reprezintă fizic microoperația de incrementare. Prin aceasta, valoarea următoare care se încarcă în registru este egală cu valoarea anterioară plus 1.

Pentru realizarea unui registru de incrementare se poate folosi structura unei numărătoare binare sau se poate construi un CLC de incrementare.

Dacă se efectuează o microoperație de incrementare pe n biți atunci circuitul logic combinațional de incrementare este un sumator format din n semisumatoare, dintre care cel corespunzător rangului zero are o intrare egală cu 1.

Pentru CLC de incrementare pe 4 biți reprezentarea este:



Rezumat.

Considerând cunoscute structurile de circuite introduse anterior, în acest paragraf sunt descrise structurile interne la nivel de poartă logică pentru principalii regiştri de microoperație.

Pentru fiecare astfel de registru se va insista în mod special pe înțelegerea modului de funcționare <u>ca circuit secvențial</u>. De asemenea, un pas important în înțelegerea acestor modele constă în identificarea rolului fiecărui circuit elementar în ansamblul constructiv al unui registru şi, în plus, studentul trebuie să poată răspunde concret, în fiecare caz în parte, la întrebări de genul: care este rolul în circuit a fiecărei componente, se poate

înlocui o componentă internă cu o alta, care este câştigul/pierderea în cazul unei astfel de înlocuiri ş.a.m.d.

Cuvinte cheie.

registru
microoperație (încărcare, deplasare, numărare, incrementare)

NOTIȚE

Anca Vasilescu
vasilex@info.unitbv.ro

2.3. Structura de bază a unui calculator

Un calculator minimal constă din 8 regiştri de bază, o unitate de memorie şi o unitate de contorizare şi control. Intre aceste componente trebuie să existe linii de comunicare, care formează **magistrale**.

In acest paragraf vom lua în discuție aceste componente, mai puțin unitatea de memorie care a fost prezentată anterior.

2.3.1. Regiştrii principali ai unui sistem de calcul

In sistemul de calcul, instrucțiunile de executat sunt memorate în locații succesive şi sunt executate secvențial. Există o unitate de control care citeşte o instrucțiune de la o adresă precizată din memorie, o execută, apoi citeşte următoarea instrucțiune, o execută, ş.a.m.d. Pentru o astfel de prelucrare, trebuie să existe:

1. o unitate de calcul care să determine în fiecare moment adresa de memorie a următoarei instrucțiuni de executat;
2. o unitate care să memoreze codul sursă al instrucțiunii citite;
3. regiştri de manipulare a datelor;
4. regiştri de memorare a diferitelor adrese care intervin în program.

Pentru aceste funcții, regiştrii unui SC pe 16 biți sunt:

Nr crt	Simbol	Nr. de biți	Nume	Funcție
1	**DR**	16	Data register	Reține un operand memorat
2	**AR**	12	Address register	Reține o adresă de memorie
3	**AC**	16	Accumulator	Registru de prelucrare
4	**IR**	16	Instruction register	Reține cod de instrucțiune
5	**PC**	12	Program counter	Reține adresa unei instrucțiuni
6	**TR**	16	Temporary register	Reține date intermediare
7	**INPR**	8	Input register	Reține un caracter de intrare
8	**OUTR**	8	Output register	Reține un caracter de ieşire

Deoarece registrul **DR** are $16 = 2^4$ biți rezultă că în acest registru se vor putea reprezenta 2^{16} combinații binare. Lungimea lui DR este egală cu lungimea cuvântului sistemului de calcul. Rezultă că acest registru recunoaşte $2^{16} : 2^4 = 2^{12} = 4096$ cuvinte (1 cuvânt = 2 octeți = 16 biți = 2^4 biți).

Registrul **AR** are 12 biți deoarece pe atât se reprezintă adresele de memorie.

Registrul **PC** reține adresa următoarei instrucțiuni care trebuie citită din memorie după ce instrucțiunea curentă va fi executată.

2.3.2. Unitatea de contorizare şi control

Cronometrarea pentru toate registrele sistemului este asigurată de o unitate (ceas) care generează impulsuri de tact.

Un impuls de ceas nu schimbă starea unui registru decât dacă aceasta este o operație permisă conform cu un semnal de control. Acesta este generat de **unitatea de control** a sistemului de calcul.

Există două tipuri de organizare a controlului: control bazat pe circuite şi control microprogramat. Prima variantă presupune implementarea semnalelor de control prin porți logice, bistabili, decodoare, ş.a. Fiecare comandă de control se regăseşte în sistem prin starea circuitelor pe care le traversează. In organizarea microprogramată, informația de control este stocată în memorie. Această zonă de memorie este programată să inițializeze secvența de microoperații de executat. Fiecare comandă de control înseamnă o actualizare a programului memorat.

Funcția de control care specifică o microoperație este o variabilă binară, numită **variabilă de control**. Una din valorile binare va corespunde executării microoperației asociate, iar cealaltă valoare nu modifică starea regiştrilor din sistem. Intr-un sistem bazat pe comunicare pe magistrale, semnalele de control care dau microoperațiile sunt grupuri de biți care selectează intrările în MUX, decodoare şi în componentele UAL.

Unitatea de control inițializează executarea secvențială pentru seturi de microoperații. La un moment dat, anumite microoperații au fost deja inițializate, în timp ce altele au rămas inactive.

In orice moment, valorile curente ale variabilelor de control se reprezintă printr-un şir de biți, numit **cuvânt de control**. Astfel, cuvintele de control pot fi programate să execute diferite operații asupra componentelor sistemului.

O unitate de control ale cărei variabile binare sunt stocate în memorie este o **unitate de control microprogramată**. Zona de memorie care păstrează variabilele de control se referă prin **memorie de control.** Fiecare cuvânt din memoria de control reprezintă o **microinstrucțiune**. O microinstrucțiune reuneşte una sau mai multe microoperații recunoscute în sistem. O secvență de microinstrucțiuni formează un **microprogram**.

Dacă, odată ce unitatea de control este activată, microprogramul nu se mai modifică atunci memoria de control poate fi o memorie de tip ROM. In acest caz, conținutul cuvântului de la o anumită adresă din această zonă de memorie reprezintă o microinstrucțiune.

2.3.3. Transferul datelor pe magistrală

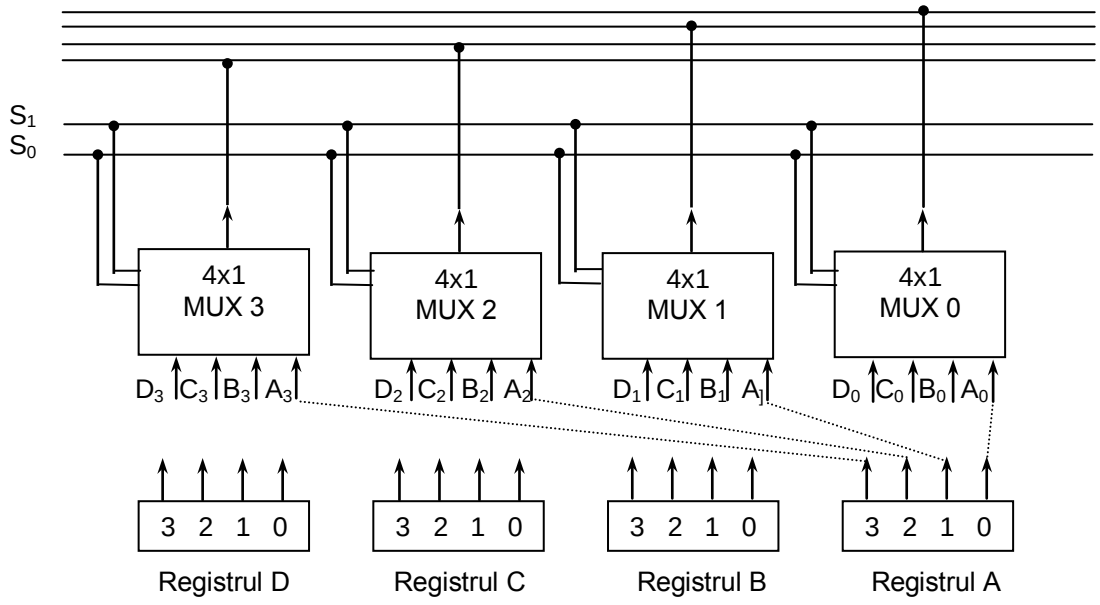
Un calculator digital trebuie să fie prevăzut cu căi de transfer pentru informațiile memorate în regiştri. Pentru aceasta, o schemă eficientă este sistemul cu **magistrală comună.**

O structură de magistrală constă într-un set de linii comune, câte una pentru fiecare bit al unui registru, prin care informația binară este transferată

la momente de timp predefinite. Pentru fiecare transfer, se selectează un registru prin semnale de control depuse pe magistrală.

O modalitate de a construi un sistem cu magistrală comună foloseşte multiplexoare. Aceştia selecteză informația din registrul sursă, care este apoi depusă pe magistrală.

Reprezentarea acestui sistem este dată în figura următoare:



Se observă că MUX0 multiplexează cei patru biți 0 ai regiştrilor, MUX1, biții 1, MUX2 biții 2 şi MUX3, biții 3. Ieşirile celor patru multiplexoare folosite formează **magistrala comună cu patru linii de date**.

In funcție de combinațiile pe intrările de selecție, pe magistrală se depune conținutul unui registru sau al altuia astfel:

S1	S0	Register selected
0	0	A
0	1	B
1	0	C
1	1	D

Rezumat.

In acest paragraf sunt enumerați principalii regiştrii care intră în structura unui calculator elementar. Detalierea unor aspecte legate de structura şi rolul în sistem a acestor regiştrii va face obiectul capitolului al treilea şi al patrulea.

Partea centrală a acestui paragraf o constituie exemplul de transfer de date din regiştri pe magistrală. Pentru acest exemplu studenții trebuie să poată justifica structura fiecărei componente (MUX, registru) folosite.

NOTIȚE

CAPITOLUL 3

Unitatea centrală de prelucrare

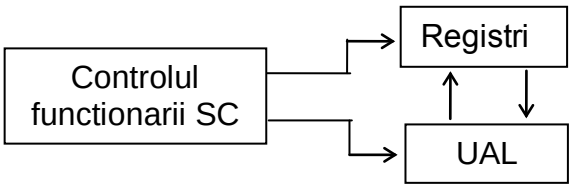
CUPRINS

CAPITOLUL 3 ---------- **85**

3.1. TRANSFERUL DATELOR (ENGL., *DATAPATH*) ---------- **87**
3.2. ORGANIZAREA STIVEI ---------- **91**
3.3. MICROOPERAȚIE. INSTRUCȚIUNE. PROGRAM ---------- **95**
3.4. CODURILE INSTRUCȚIUNILOR ---------- **96**
3.5. MODURI DE ADRESARE ---------- **100**
3.6. CLASIFICAREA INSTRUCȚIUNILOR ---------- **102**
3.6.1. Instrucțiuni pentru transferul datelor ---------- *104*
3.6.2. Instrucțiuni pentru manipularea datelor ---------- *104*
3.6.3. Instrucțiuni pentru controlul programului ---------- *105*
3.7. PRELUCRAREA INSTRUCȚIUNILOR UNUI PROGRAM ---------- **111**
3.8. MICROPROCESORUL ---------- **111**
3.8.1. Caracteristici ---------- *112*
3.8.2. Setul de instrucțiuni ale microprocesorului ---------- *114*
3.8.3. Modelarea funcționării unui procesor ---------- *115*
3.8.3.1. Maşina cu trei adrese ---------- 115
3.8.3.2. Maşina cu o adresă ---------- 118
3.8.4. Structura internă de bază a unui microprocesor I 80x86 ---------- *122*
3.9. COPROCESOARE MATEMATICE ---------- **125**
3.10. MULTIPROCESARE ---------- **125**

Componenta calculatorului care efectuează cea mai mare parte a operațiilor de prelucrare a datelor este numită **unitatea centrală de prelucrare**, prescurtat UCP (în engleză, CPU).

UCP constă din trei părți fundamentale, reprezentate în figura următoare:



Regiştrii memorează datele intermediare folosite în timpul executării instrucțiunilor. Unitatea aritmetico – logică, UAL, execută microoperațiile solicitate. Unitatea de control supervizează transferul informațiilor între regiştrii şi anunță UAL asupra următoarei operații de efectuat.

In acest capitol vom descrie modul în care comunică regiştrii cu UAL prin intermediul liniilor de magistrală şi vom explica mecanismul de lucru cu stiva. Apoi, vom descrie tipuri pentru formatele instrucțiunilor, moduri de acces la memorie şi instrucțiuni care trebuiesc să existe în setul minimal al oricărui SC. In final, vom fi în măsură să abordăm componenta fizică fundamentală a unui calculator, procesorul.

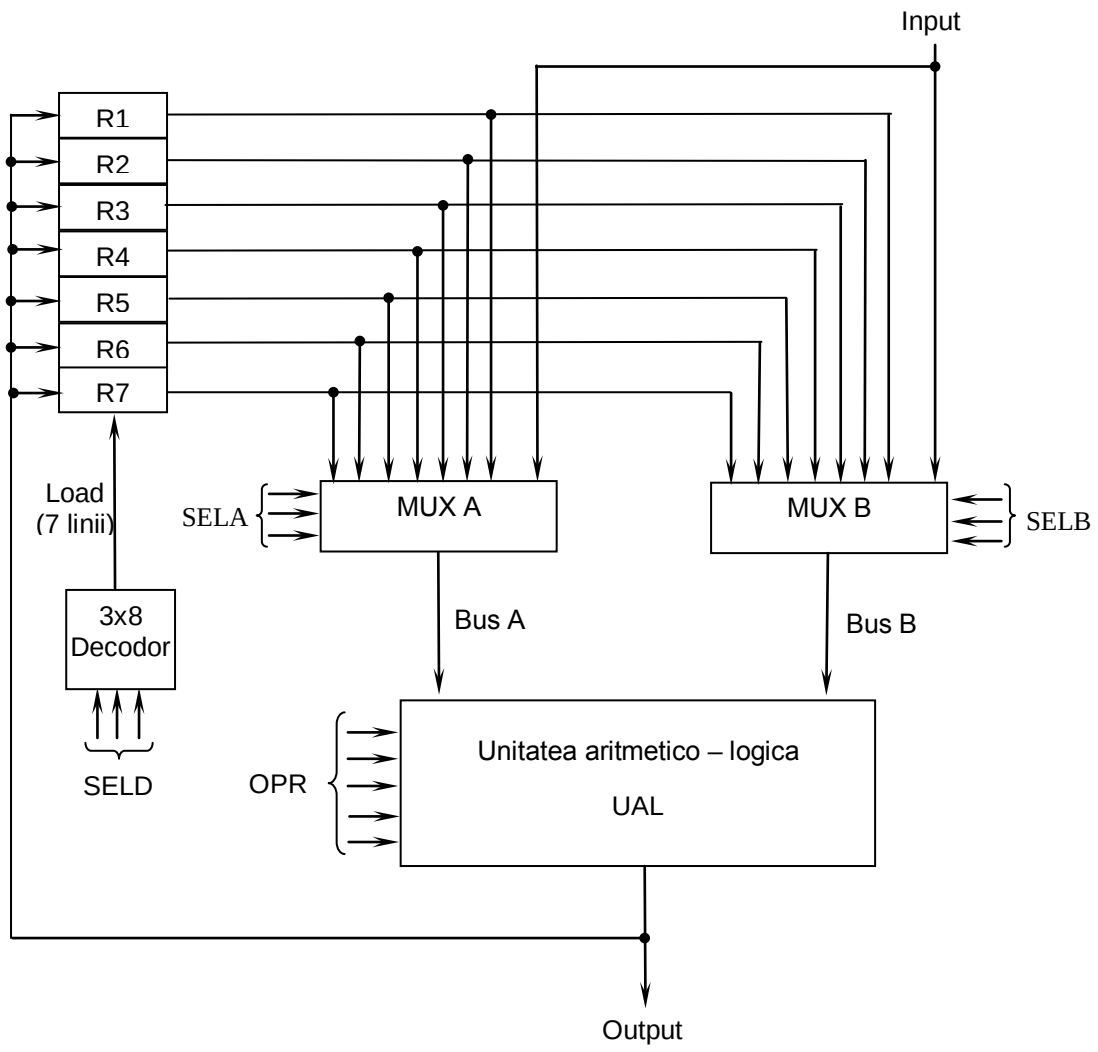
3.1. Transferul datelor (engl., *datapath*)

Locațiile de memorie sunt necesare pentru a stoca pointeri, rezultate ale diferitelor numărări, adrese de revenire, rezultate intermediare, produse parțiale, ş.a.m.d. Pentru a scurta timpul de acces la zonele de memorie care păstrează aceste date, practic, acestea se memorează în **regiştrii de prelucrare** (*processor registers*). Când în UCP există un număr mare de regiştri, se recomandă ca aceştia să fie conectați prin linii de magistrală, de tip *common bus system*.

Organizarea unei magistrale pentru şapte regiştri UCP (conectați la UAL comună) ar putea fi cea de pe pagina următoare.

Ieşirea din fiecare registru este conectată la două MUX pentru a forma cele două magistrale A şi B. Liniile de adresă ale fiecărui MUX selectează un registru sau linia de date pentru una dintre magistrale. Datele de pe magistralele A şi B reprezintă intrările în UAL comună. Operația selectată în UAL determină tipul microoperației care se va efectua. Rezultatul microoperației este disponibil pentru linia de date de ieşire, care este legată la intrările tuturor regiştrilor. Un decodor selectează registrul care va prelua informația de pe *Output* şi activează intrarea de control *Load* pentru preluarea datelor în acel registru. Apoi, asigură transferul de date între *Output* şi intrarea în registru destinație selectat.

Notăm: SELA – intrarea de selecție pentru MUX A, SELB – intrarea de selcție pentru MUX B, OPR – selectorul de operație pentru UAL şi SELD – selectorul registrului destinație pentru decodor.



In acest exemplu avem 14 intrări de selecție în circuit: 3 pentru SELA, 3 pentru SELB, 3 pentru SELD şi 5 pentru OPR. Combinația valorilor pe aceste intrări formează **cuvântul de control** (*control word*). Acesta se reprezintă astfel:

3	3	3	5
SELA	SELB	SELD	OPR

Cei trei biți de pe intrările SELA selectează registrul sursă pentru intrarea A a UAL, iar SELB selectează registrul sursă pentru intrarea B a UAL. Cei trei biți ai SELD selectează registrul destinație folosind decodorul şi cele 7 ieşiri de validare a încărcării, *Load*. In concluzie, cuvântul de control aplicat pe intrările de selecție precizează o microoperație, conform cu următoarele tabele:

	Registrul selectat de		
Conținutul pentru	SELA	SELB	SELD
000	Input	Input	nimic
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

Conținutul pentru OPR	Operația	Simbolul
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	Add A + B	ADD
00101	Substract A – B	SUB
00110	Decrement A	DECA
01000	AND A şi B	AND
01010	OR A şi B	OR
01100	XOR A şi B	XOR
01110	Complement A	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA

Rezumat.

Acest paragraf conține un exemplu de *datapath* (cale a datelor) pentru comunicare între UAL şi regiştri. Scopul acestui exemplu este dublu: pe de o parte să familiarizeze studenții cu un ansamblu complex de componente logice care include şi linii de comunicare (magistrale) şi, pe de altă parte, să exemplifice conceptul de **cuvânt de control** (structură şi utilizare). Acest al doilea obiectiv este important în perspectiva înțelegerii *codului de instrucțiune*, respectiv a formatului de descriere a programului de executat la nivelul procesorului.

Verificare.

1. Pentru exemplul de cale a datelor (engl. *datapath*) prezentat anterior, răspundeți la următoarele cerințe:

a) justificați tipul decodorului 3x8;

b) cum se modifică structura de date dacă UAL reprezintă codul operațiilor pe 7 biți?

c) este posibil să conectăm la o astfel de magistrală 3 regiştri; cum se modifică dimensiunile componentelor logice în acest caz?

d) dimensiunea regiştrilor este determinată? Depind alte dimensiuni de dimensiunea regiştrilor?

2. Pe modelul *datapath* anterior, se cere:

a) să se aleagă valori concrete pentru conținutul celor şapte regiştri pe patru biți;

b) să se execute următorul program de control:

01010001100010
10001111101100
00111000110000
11100111100001

NOTIȚE

Anca Vasilescu
vasilex@info.unitbv.ro

3.2. Organizarea stivei

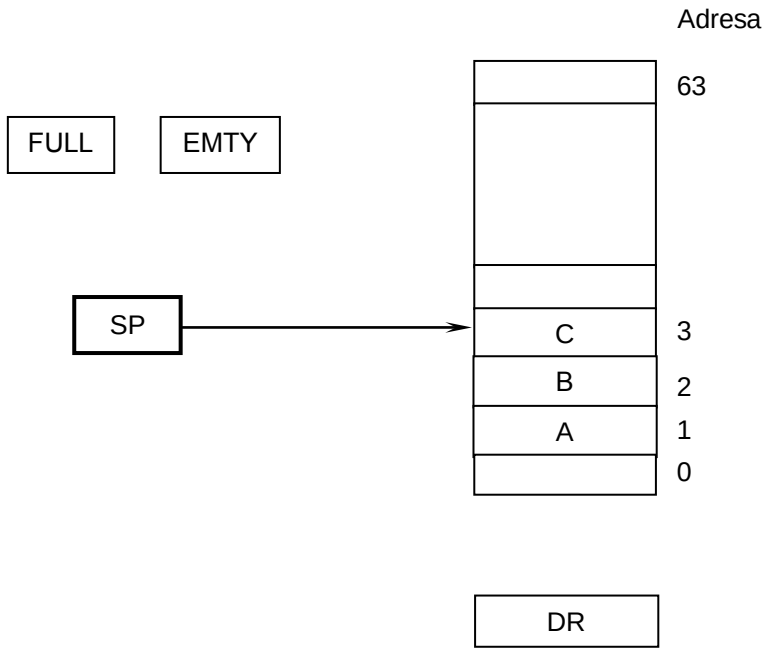
O structură importantă care este inclusă în unitatea centrală de prelucrare la majoritatea SC este cea de stivă sau listă de tip LIFO. Aici, prin **stivă** vom înțelege o unitate de memorare care se manipulează astfel încât ultima dată introdusă este prima care se prelucrează.

Intr-un sistem de calcul digital, stiva este în esență o unitate de memorie cu un registru de adresă care execută o singură operație, anume cea de contorizare (numărare), după ce a fost încărcat cu o valoare inițială. Registrul care păstrează adresa pentru stivă se numeşte **pointer de stivă** (*stack pointer*, **SP**). Valoarea acestuia întotdeauna *pointează* spre entitatea încărcată în vârful stivei.

Operațiile care se pot efectua asupra unei stive sunt cele de **inserare** şi **ştergere**. Inserarea se mai numeşte *push*, iar ştergerea, *pop*.

In stiva unui SC nu se inserează şi nu se şterge nimic. Aceste operații sunt simulate prin incrementarea şi decrementarea valorii din registrul SP.

Stiva SC poate fi depusă într-o zonă contiguă de memorie sau poate fi organizată ca o colecție finită de cuvinte de memorie sau de regiştri. Pentru un registru de stivă cu 64 cuvinte de memorie, reprezentarea schematică este:



Registrul SP conține o valoare binară egală cu adresa cuvântului care este la momentul respectiv în vârful stivei.

In acest exemplu avem trei entități în stivă: A, B şi C. C este în vârful stivei, astfel că în SP avem valoarea 3.

Pentru a şterge vârful stivei, se efectuează o operație *pop* pe stivă prin citirea cuvântului de memorie de la adresa 3 şi decrementarea conținutului lui SP. Astfel, SP memorează valoarea 2, adică pointează la B şi conținutul B este considerat în vârful stivei. Practic, C a fost citit, nu mai este în vârful stivei, dar nu a fost eliminat din stivă.

Pentru a insera un element în stivă se efectuează o operație *push* prin incrementarea valorii lui SP şi scrierea cuvântului la următoarea adresă liberă.

Pentru o stivă de 64 cuvinte de memorie, SP are 6 biți deoarece $64 = 2^6$. Astfel, SP nu poate memora o valoare mai mare decât 63 = 111111. Când 63 este mărit cu o unitate rezultatul este 0 deoarece

111111 + 1 = 1000000

şi SP reține biții cel mai puțin semnificativi, adică 000000.

Analog, când 000000 este decrementat cu 1, rezultatul este 111111.

Pentru analizarea acestor situații particulare, se folosesc doi regiştri speciali, fiecare de 1 bit, numiți FULL şi EMTY. Registrul FULL este setat pe 1 când stiva este plină, iar EMTY este setat pe 1 când stiva este vidă.

Registrul de date DR reține ceea ce se va scrie în stivă la următoarea operație *push* sau ceea ce s-a citit din stivă la ultima operație *pop*.

Funcționarea stivei constă în precizarea operațiilor elementare pentru *push* şi *pop*.

Inițial, SP este şters, deci are valoare 0, EMTY este setat pe 1 şi FULL este setat pe 0. Astfel, SP pointează cuvântul de la adresa 0.

Dacă stiva nu este plină atunci este permisă o operație *push* prin:

(SP) <— (SP) + 1

[SP] <— (DR)

IF (SP) = 0 THEN FULL <— 1

EMTY <— 0

Dacă stiva nu este goală atunci este permisă o operație *pop* prin:

(DR) <— [SP]

(SP) <— (SP) – 1

IF (SP) = 0 THEN EMTY <— 1

FULL <— 0

Observație. Prin SP ne referim la dresa registrului SP, prin (SP) ne referim la conținutul registrului SP, iar prin [SP] ne referim la conținutul de la adresa conținută de registrul SP.

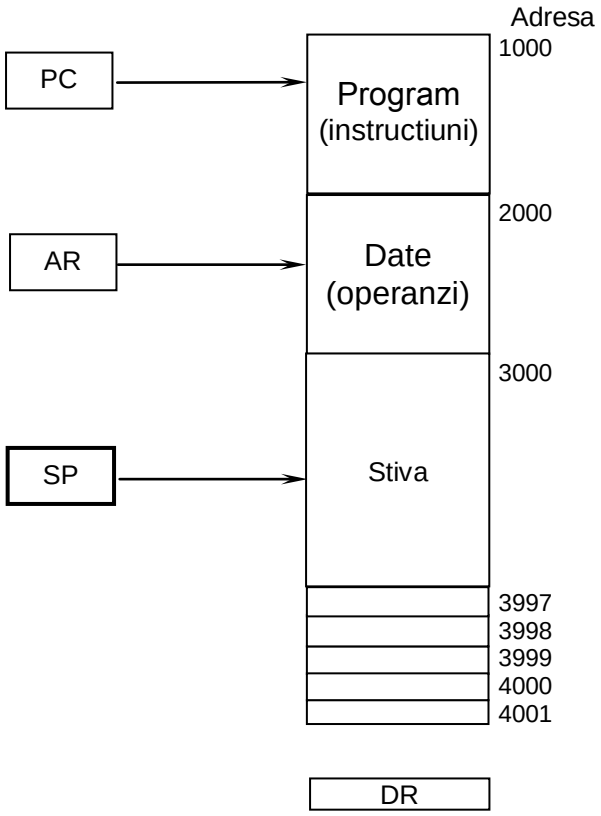
Memoria rezervată pentru stivă poate fi o unitate individuală sau poate fi implementată într-un RAM ataşat UCP. Implementarea unei stive în UCP se face prin asignarea unei zone de memorie pentru operațiile pe stivă şi prin folosirea unui registru de prelucrare care să îndeplinească funcțiile lui SP.

In reprezentarea de pe pagina următoare memoria SC este împărțită în trei segmente: segment de program, segment de date şi stiva.

Registrul PC, *Program Counter*, pointează la adresa următoarei instrucțiuni din programul de executat. Registrul de adresă AR pointează un şir de date. SP pointează spre vârful stivei. Cei trei regiştri sunt conectați la o magistrală comună de adrese şi numai unul poate depune o adresă în memorie.

Registrul PC este folosit pentru a citi o instrucțiune de executat. AR este folosit în execuție pentru a citi un operand. SP este folosit pentru a simula inserarea sau ştergerea unui element din stivă.

Cele mai multe SC nu au componente hardware care să verifice situațiile de operare pe stivă în configurații improprii, cum ar fi depăşirile de tip **overflow**, pentru stivă plină şi **underflow**, pentru stivă vidă. Limitele stivei pot fi verificate prin mecanisme soft, de exemplu prin folosirea a doi regiştri de prelucrare: unul va memora adresa cea mai mică a cuvintelor de memorie rezervate stivei, iar celălalt adresa cea mai mare. Cu acestea, după o operație *push*, SP se va compara cu primul registru, iar după o operație *pop*, cu valoarea din al doilea registru.



Observație. In exemplul stivei pe 64 cuvinte de memorie, o inserare în stivă corespunde unei operații de mărire a adresei la care se încarcă conținutul lui DR. In exemplul următor, în care am reprezentat stiva pe o zonă de memorie de la adresa 4001 la 3000, o inserare în stivă corespunde unei micşorări a adresei de memorie la care se încarcă conținutul registrului DR. De acest aspect trebuie ținut cont în scrierea instrucțiunilor care asigură utilizarea stivei, începând chiar cu valoarea cu care se inițializează SP. Aceasta trebuie să fie adresa rezervată zonei de memorie de la baza stivei.

Avantajul lucrului cu stiva constă în faptul că UCP poate referi memoria stivei fără să fie nevoie să precizeze o anumită adresă, deoarece adresa respectivă este întotdeauna disponibilă şi este automat actualizată în registrul de stivă, SP.

Rezumat.

Stiva, ca structură de date, face parte din tematica cursurilor de programare. In ceea ce priveşte arhitectura calculatoarelor, interesează stiva sistemului de calcul ca zonă de memorie rezervată şi organizată pe principiul stivei (LIFO). De asemenea, ca şi în cazul paragrafului precedent, conceptul este esențial pentru înțelegerea modului în care procesorul (implicit, programatorul prin programul pe care îl scrie în limbaj de asamblare) lucrează cu cele trei segmente de memorie: stivă, date, cod.

In particular, pentru lucru cu segmentul de stivă, trebuie reținut rolul registrului SP în efectuarea operațiilor pe stivă (încărcare, descărcare).

Cuvinte cheie.

stiva sistemului de calcul
registrul SP, operații specifice: încărcare, descărcare
eroare de depăşire

Verificare.

1. Dacă stiva de memorie a unui sistem de calcul are 256 de cuvinte, care trebuie să fie capacitatea minimă a registrului SP în acest caz?
2. Ce se poate spune despre rezultatul unei operații *push* pe stivă dacă în urma efectuării ei conținutul registrului SP este nul?

NOTIȚE

3.3. Microoperație. Instrucțiune. Program

Organizarea hardware internă a unui SC este bine definită dacă se specifică:

1. mulțimea de regiştri şi funcțiile lor;
2. secvența de microoperații de executat pentru informațiile binare depuse în fiecare registru;
3. instrucțiunea care inițializează fiecare secvență de microoperații.

Pentru specificarea secvențelor de operații de transfer de informații între regiştri se foloseşte un simbolism, numit **limbaj de transfer pe regiştri** (*register transfer language*).

Un SC digital este capabil să execute diferite microoperații şi, în plus, poate primi instrucțiuni pentru precizarea următoarei secvențe de microoperații care trebuie executată.

Pentru SC, o **instrucțiune** este un cod binar care precizează o secvență de microoperații de executat de către calculator.

Un **program** este un set de instrucțiuni care precizează operațiile, operanzii şi legăturile dintre aceste elemente în execuție.

Există o mare varietate de limbaje de programare pe care utilizatorul le poate folosi pentru a scrie un program pentru calculator, în timp ce sistemul de calcul poate executa un program numai dacă este reprezentat intern în binar. De aceea, un program scris în orice limbaj de programare trebuie întâi *translatat* (tradus) în reprezentare binară.

Programele scrise pentru un calculator se pot încadra într-una dintre următoarele categorii:

1. **cod binar**, dat de o secvență de instrucțiuni şi operanzi binari care reproduce exact modul de reprezentare în memorie a secvenței respective. Un astfel de program se va numi **program în cod maşină** sau **program în limbaj maşină**;
2. **cod octal** sau **hexazecimal**, dat de o conversie a codului binar în octal sau hexa;
3. **cod simbolic**, dat de o reprezentare cu simboluri a codurilor instrucțiunilor. Fiecare instrucțiune dată prin simboluri poate fi translatată
   într-o instrucțiune în cod binar. Această translatare este făcută de un program special, numit **assembler**. Deoarece un *assembler* traduce simboluri, aceste programe simbolice se numesc *programe în limbaj de asamblare*;
4. **limbaje de programare de nivel înalt**. Un program care traduce un astfel de program în binar se numeşte **compilator**.

Rezumat.

Acest paragraf defineşte conceptele de **instrucțiune** şi **program**, în perspectiva paragrafelor ulterioare şi, în plus, pentru înțelegerea noțiunii de **cod** – ca mod de reprezentare a unui program. Astfel, în funcție de cunoştințele şi de preferințele programatorului, acesta poate opta pentru una dintre variantele de scriere a codului sursă al programului său: cod binar, cod octal, cod simbolic sau cod în limbaj de programare de nivel înalt.

3.4. Codurile instrucțiunilor

Pentru un SC cu control programat, calculatorul citeşte fiecare instrucțiune din memorie şi o depune într-un *registru de control*. Unitatea de control interpretează acest cod şi determină execuția lui, prin inițializarea unei secvențe de microoperații.

Fiecare calculator posedă un set propriu de instrucțiuni pe care le poate executa. Capacitatea de a memora şi de a executa instrucțiunile, precum şi conceptul de *program memorat* sunt proprietățile principale ale unui calculator de uz general.

Codul unei instrucțiuni este dat de un grup de biți care dirijează calculatorul pentru a executa o anumită operație. De obicei, acesta este delimitat în două câmpuri, fiecare având o semnificație bine precizată.

Cea mai mare parte a codului formează zona operațională, numită **codul operației** (*operation code*). Acesta defineşte tipul operației, de exemplu: adunare, scădere, înmulțire, deplasare, complementare. Numărul de biți rezervați pentru codul operației depinde de numărul total de instrucțiuni ale SC analizat.

Observație. Este bine să precizăm aici care este diferența între o operație (*computer operation*) şi o microoperație. O operație este o parte dintr-o instrucțiune stocată în memorie. Ea este dată de un cod binar care o identifică în sistemul de calcul. Unitatea de control preia instrucțiunea din memorie şi îi interpretează biții care reprezintă codul operației. Apoi, generează o secvență de semnale de control pentru a inițializa microoperațiile corespunzătoare în regiştri interni ai SC. Pentru fiecare cod de operație, unitatea de control generează o secvență de microoperații necesare executării operației specificate. De aici, codul operației se numeşte de multe ori **macrooperație** deoarece specifică un set de microoperații.

Revenind, codul operației reprezintă operația de executat. Această operație trebuie să lucreze cu date memorate în regiştrii de prelucrare sau direct în memorie. De aceea, codul unei instrucțiuni trebuie să precizeze, nu numai operația, ci şi informații cu privire la operanzi. Aceste informații formează **zona de adresă**, care conține regiştrii sau cuvintele de memorie unde se găsesc operanzii şi registrul sau cuvântul de memorie în care se depune rezultatul. Registrele de prelucrare se regăsesc prin încărcarea în zona de adresă a unui cod binar de k biți care să identifice în mod unic unul dintre cei 2^k regiştri din sistem. Cuvintele de memorie se dau în codul instrucțiunii prin adresele lor.

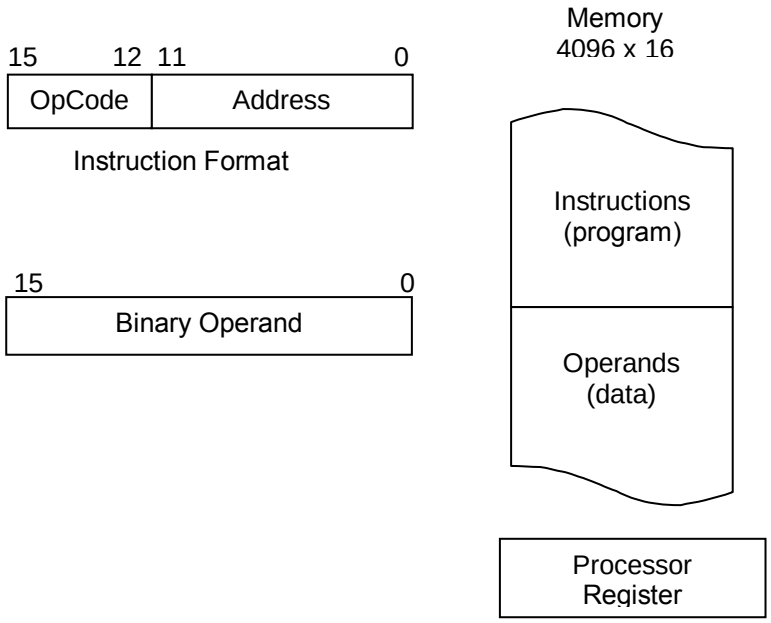
Există numeroase modalități de a organiza codul binar al unei instrucțiuni. Fiecare SC are propriul format al codului de instrucțiune. Acesta este conceput de designer-ul calculatorului, adică de cel care precizează arhitectura SC.

In continuare dăm trei exemple de organizare a formatului instrucțiunilor, construite mai mult pe considerente didactice decât cu referire la anumite sisteme de calcul.

Exemplul 1.

Cel mai simplu mod de a organiza un calculator este de a folosi un registru de prelucrare şi un cod de instrucțiune format din *OpCode* şi o adresă. Această adresă de memorie indică unității de control unde găseşte operandul pe care trebuie să-l folosească. Acest operand va fi citit din memorie şi valoarea rezultată va fi operată cu conținutul registrului de prelucrare.

Acest tip de organizare este prezentat în figura următoare:



Instrucțiunile sunt depuse într-o zonă de memorie distinctă față de cea în care sunt depuse datele. Aici avem o memorie cu $4096 = 2^{12}$ cuvinte, deci adresele lor se vor reprezenta pe 12 biți.

Dacă codul unei instrucțiuni se reprezintă pe 16 biți atunci rezervăm 12 biți pentru adresa unui operand şi rămân 4 biți pentru codul operației, *OpCode*. Astfel, acest sistem va recunoşte un set de $16 = 2^4$ operații.

Unitatea de control citeşte cei 16 biți ai codului unei instrucțiuni din zona de memorie rezervată programului. Cei 12 biți ai zonei de adresă a codului identifică în zona de date locația din care să se citească operandul pe

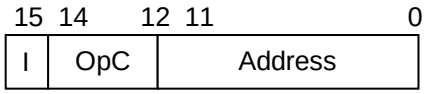
16 biți necesar. Apoi, unitatea de control inițializează operația specificată de cei 4 biți ai *OpCode*. Aceasta se execută pe actualul conținut al registrului de prelucrare (notat aici cu AC) şi pe valoarea operandului citit din memorie.

Observație. Uneori este util să folosim zona de adresă a unei instrucțiuni nu cu semnificația unei adrese ci ca valoare a operandului de citit. Când partea a doua a unui cod de instrucțiune reprezintă un operand, spunem că instrucțiunea are **operand transmis prin valoare**. Când zona de adresă a unui cod de instrucțiune conține adresa la care se găseşte în memorie operandul necesar, spunem că **operandul este transmis prin adresă** (directă).

O a treia posibilitate este ca la adresa din zona de adresă a codului instrucțiunii să se găsească adresa de memorie a operandului. In acest caz, **operandul este transmis prin adresă indirectă**.

Exemplul 2.

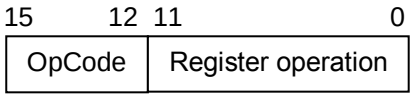
Pentru a face diferența între adresele directe şi indirecte se poate rezerva în formatul instrucțiunii un bit modal (*de mod*) care va avea o anumită valoare binară dacă adresa este indirectă şi valoarea complementară dacă adresa este directă. Acesta se notează de obicei cu I (*indirect address mode bit*) şi se reprezintă ca în figura următoare:



Exemplul 3.

Dacă o operație nu are nevoie de operand din memorie atunci zona de adresă din codul instrucțiunii poate fi folosită în alte scopuri, de exemplu pentru a preciza operații suplimentare recunoscute de sistem. Se obține astfel o instrucțiune cu referință la un registru de prelucrare (*register – reference instruction*).

In acest caz, în zona de adresă a codulului instrucțiunii nu mai avem o adresă ci o operație sau un test de efectuat asupra registrului de prelucrare.



Rezumat.

Conținutul acestui paragraf este central în contextul capitolului al treilea. Aici se defineşte **codul de instrucțiune** cu cele două câmpuri ale sale: zona de cod al operației şi zona de adresă.

Exemplele care se dau în acest paragraf sunt esențiale pentru înțelegerea modurilor de adresare. Astfel, fiecare exemplu reprezintă o situație particulară, concretă, pentru combinațiile de valori (semnificații) pe care le pot avea cele două zone ale codului de instrucțiune: zona de cod şi zona de adresă.

De asemenea, exemplele din acest paragraf anticipează formatul de instrucțiune folosit de maşina cu trei adrese, respectiv de maşina cu o adresă (vezi *Modelarea funcționării unui procesor*).

Verificare.

1. Pentru exemplul 1 anterior, care este lungimea minimă a cuvântului pentru o memorie de 1024 cuvinte?
2. Refaceți calculele de la exemplul 1 în cazul unei memorii de 64 cuvinte de 8 biți fiecare. Câte operații se pot defini pe o astfel de structură?

NOTIȚE

3.5. Moduri de adresare

In codul unei instrucțiuni, OpCode specifică operația de executat. Această operație trebuie executată asupra unei date salvate în memorie sau într-un registru. Felul în care este regăsit operandul depinde de conținutul zonei de adresă a codului instrucțiunii, deci de **modul de adresare** pentru fiecare instrucțiune. Acesta specifică o regulă pentru interpretarea şi modificarea câmpului adresă a codului instrucțiunii înainte ca operandul să fie referit.

Pentru a înțelege modurile de adresare trebuie cunoscută secvența de operații pe care le efectuează unitatea de control pentru fiecare instrucțiune în parte. Aceasta constă din următoarele etape:

1. preluarea instrucțiunii din memorie (faza de **extragere**, engl., *fetch*),
2. decodificarea instrucțiunii (faza de **decodificare**, engl., *decode*),
3. executarea instrucțiunii (faza de **execuție**, engl., *execute*).

In calculator există un registru special, PC, care păstrează *urma* instrucțiunilor programului memorat. PC reține adresa instrucțiunii care se va executa şi este incrementat la fiecare preluare a unei instrucțiuni din memorie.

Etapa de decodificare identifică operația de executat, modul de adresare a instrucțiunii şi localizează operandul.

Apoi, calculatorul execută instrucțiunea respectivă şi revine la pasul 1 pentru a prelua următoarea instrucțiune de executat.

Modul de adresare se poate regăsi în codul instrucțiunii pe un bit sau pe mai mulți. Intr-un sistem de calcul se poate ca modul de adresare să difere de la o instrucțiune la alta sau, chiar pentru o aceeaşi instrucțiune adresarea să fie combinată.

A. Pentru regăsirea operandului, **există două moduri de adresare care nu folosesc o adresă în câmpul al doilea al codului instrucțiunii**. Acestea sunt **modul implicit** şi **modul direct**.

In modul implicit de adresare se presupune că operanzii sunt dați prin definiția instrucțiunii, fără ambiguitate. In modul direct de adresare, operandul apare în codul instrucțiunii, în câmpul de adresă. Un exemplu de astfel de instrucțiuni sunt cele de inițializare a conținutului unui registru cu o anumită valoare.

B. Când câmpul de adresă al codului de instrucțiune specifică un registru de prelucrare, spunem că instrucțiunea este în mod registru. In acest caz operandul se află în registrul specificat şi acesta va fi selectat din lista de identificare a regiştrilor sistemului.

Un caz particular de adresare cu registru este situația în care în registrul referit se află adresa cuvântului de memorie care reține operandul căutat. Spunem atunci că **adresarea este indirectă**.

Particularizând în continuare, putem avea o instrucțiune care incrementează (decrementează) valoarea din registru la fiecare accesare a memoriei. In acest caz avem **adresare indirectă în mod registru cu**

autoincrementare (autodecrementare). Un exemplu pentru această situație sunt regiştrii care referă o tabelă de date depusă în memorie.

Câmpul de adresă al codului de instrucțiune este folosit de unitatea de control pentru a obține valoarea operandului memorat. Uneori, valoarea dată în acest câmp este adresa operandului (vezi situațiile anterioare), dar poate fi şi adresa la care se determină (calculează) adresa operandului. Spunem că avem **adresa absolută a operandului** dacă ea reprezintă adresa de memorie la care se găseşte operandul căutat.

In exemplele anterioare am anticipat modul de adresare directă şi modul de adresare indirectă. In concluzie, putem spune că la adresarea directă adresa efectivă a operandului apare în câmpul de adresă al instrucțiunii. La adresarea indirectă, câmpul de adresă al instrucțiunii cuprinde adresa la care este stocată în memorie adresa efectivă a operandului.

C. In alte situații, **modul de adresare impune ca valoarea memorată în câmpul de adresă să fie adunată cu conținutul unui anumit registru pentru a obține adresa efectivă a operandului căutat**. Un astfel de registru implicat în calcul de adrese poate fi PC, un **registru de index** sau un **registru de bază**.

Dacă este vorba de registrul PC atunci avem **adresare relativă**. Concret, când conținutul din câmpul de adresă este adunat cu conținutul registrului PC se obține o adresă efectivă a cărei poziție în memorie este relativă la adresa următoarei instrucțiuni de executat (reținută în PC).

Exemplu. M[PC]=825, address=24. Instrucțiunea de la locația 825 este citită din memorie şi PC <– PC + 1 = 826. Rezultă că se calculează adresa 826 + 24 = 850, care reprezintă a-24-a locație de memorie de la adresa următoarei instrucțiuni de executat.

Avantajul acestei adresări este faptul că adresa relativă se reprezintă pe o secvență de biți mai scurtă decât adresa efectivă.

Dacă pentru a obține adresa efectivă se adună conținutul câmpului de adresă cu conținutul unui registru de index atunci avem o **adresare indexată**. Un registru de index este un registru de prelucrare special care conține o valoare de ordine sau de indexare (indiciere). Aici, codul instrucțiunii cuprinde adresa de început a unui şir de date memorat şi fiecare dată din şir se regăseşte în memorie relativ la adresa de început. Diferența între adresa de început şi adresa efectivă a operandului este indexul (indicele) memorat în registrul de index.

Dacă pentru a obține adresa efectivă se adună conținutul câmpului de adresă cu conținutul unui registru de bază atunci avem o **adresare cu registru de bază**. Diferența față de adresarea indexată constă în tipul registrului implicat în calculul de adresă. Aici, registrul de bază conține o adresă de bază şi codul instrucțiunii dă deplasamentul relativ la adresa de bază respectivă.

Execuția repetată a secvenței *extrage-decodifică-execută* aplicată pe instrucțiunile unui program oarecare arată că un program poate fi executat de către un alt program, care va fi pentru primul un **interpretor**. Din acest punct de vedere execuția unui program se poate face prin hardware, caz în care

procesorul este cel care execută efectiv programmul, sau prin software, în cazul execuției prin interpretor. Această echivalență între procesoarele hardware şi interpretoare are consecințe foarte importante pentru structurarea şi proiectarea sistemelor de calcul. Astfel, după specificarea limbajului maşină L al unui nou calculator, echipa de proiectare poate decide dacă va construi un procesor hardware care să execute direct programele scrise în limbajul L sau dacă va scrie un interpretor care să interpreteze programle scrise în L (adică să le traducă în limbajul maşină al unui procesor hardware deja existent). Sunt posibile şi construcții hibride, cu o parte de execuție hardware şi o parte de interpretare software.

Extragerea instrucțiunilor din memorie este un punct critic în viteza de execuție a acestora. Pentru a rezolva această problemă, calculatoarele aveau posibilitatea de a extrage în avans instrucțiuni din memorie, astfel încât acestea să fie rapid disponibile atunci când erau necesare. Aceste instrucțiuni erau păstrate într-un set de registre numit **tamponul pentru extragere în avans**. In principiu, extragerea în avans împarte execuția instrucțiunii în două etape: extragerea şi execuția propriu-zisă. Conceptul **benzii de asamblare a procesorului** (engl. *pipeline*) extinde strategia: în loc să se împartă execuția instrucțiunii în doar două părți, de multe ori aceasta este împărțită în mai multe etape, de fiecare etapă ocupându-se o componentă hardware dedicată acesteia, toate etapele putând să fie executate în paralel. De exemplu, o bandă de asamblare cu 5 segmente (etape) ar putea fi delimitată astfel:

1. unitatea de extragere a instrucțiunii;
2. unitatea de decodificare a instrucțiunii;
3. unitatea de extragere a operanzilor;
4. unitatea de execuție a instrucțiunii;
5. unitatea de scriere a rezultatului.

Rezumat.

In funcție de conținutul concret al fiecăreia dintre zonele din codul de instrucțiune, în programul propriu-zis al procesorului, se deosebesc modurile de adresare. In acest paragraf am dat o clasificare a modurilor de adresare în trei categorii:

(A) adresare *fără adresă*

1. modul implicit
2. modul direct

(B) adresare în mod registru

1. adresare indirectă
2. adresare absolută

(C) adresare cu registru de calcul

1. adresare relativă
2. adresare indexată
3. adresare cu registru de bază.

Cuvinte cheie.

mod de adresare
banda de asamblare a procesorului

Verificare.

1. Care este diferența dintre un interpretor şi un compilator?
2. Identificați cele cinci etape ale *pipeline*-ului intern pentru un program de control scris pentru exemplul de *datapath* de la începutul capitolului.

NOTIȚE

3.6. Clasificarea instrucțiunilor

Deşi setul de instrucțiuni diferă de la un calculator la altul, toate sistemele de calcul trebuie să aibă un ansamblu minimal de instrucțiuni. Acestea se regăsesc în următoarele trei categorii:

1. instrucțiuni pentru transferul datelor – asigură transferul datelor dintr-o locație în alta, fără modificarea conținutului;
2. instrucțiuni pentru manipularea datelor – efectuează operațiile aritmetice, logice şi cele de deplasare;
3. instrucțiuni de control al programului – cu rol în luarea deciziilor şi în dirijarea execuței microprogramului.

In particular, unele sisteme de calcul au în plus instrucțiuni pentru operații de transfer pe regiştri şi / sau instrucțiuni care permit utilizatorului să intervină asupra microprogramului.

3.6.1. Instrucțiuni pentru transferul datelor

Cele mai uzuale transferuri se realizează:

✓ între memorie şi regiştrii de prelucrare (instrucțiunea **Load** sau **Store**);

✓ între regiştrii de prelucrare şi stivă (instrucțiunea **Push** sau **Pop**);

✓ între regiştrii de prelucrare şi regiştrii de intrare / ieşire (instrucțiunea **Input** sau **Output**);

✓ între doi regiştri de prelucrare (instrucțiunea **Move** sau **Exchange**).

In funcție de conținutul zonei de adresă a codului, instrucțiunea poate fi apelată cu diverşi parametri. Cunoaşterea şi utilizarea acestora țin de limbajul de asamblare folosit şi de sistemul de calcul pe care se lucrează.

3.6.2. Instrucțiuni pentru manipularea datelor

Această clasă de instrucțiuni reprezintă posibilitățile de calcul ale sistemului. Din această categorie fac parte:

a) instrucțiunile aritmetice;

b) instrucțiunile logice şi cele de prelucrare la nivel de bit;

c) instrucțiunile de deplasare.

In general, calculatoarele *cunosc* operațiile aritmetice de adunare, scădere, înmulțire şi împărțire. Este posibil ca unele sisteme să folosească numai adunarea şi înmulțirea sau chiar numai adunarea şi să simuleze şi celelalte operații. In funcție de operanzii pe care poate lucra o **instrucțiune aritmetică**, aceasta poate avea diferite forme. De exemplu, nu sunt puține calculatoarele care admit trei sau chiar mai multe instrucțiuni pentru adunare. Acestea pot fi: ADDI, pentru operanzi binari – întregi, ADDF, pentru operanzi în virgulă flotantă sau ADDD, pentru operanzi zecimali.

Instrucțiunile logice efectuează operații binare pe şiruri de biți memorați în regiştri. Fiecare bit este considerat individual şi tratat ca o variabilă booleană. Operațiile binare fundamentale sunt: setare pe 0, setare pe 1 şi complementare.

La rândul lor, **instrucțiunile de deplasare** pot fi de tip logic, de tip aritmetic sau de rotație. De exemplu, deplasarea logică este utilă atunci când trebuie completate cu 0 pozițiile libere ale unei valori care nu ocupă întregul registru. Deplasarea aritmetică asigură fixarea pe poziție corectă a bitului de semn. Rotația reprezintă o deplasare circulară, adică un bit *eliminat* la o deplasare la dreapta este încărcat în bitul cel mai semnificativ eliberat.

3.6.3. Instrucțiuni pentru controlul programului

Instrucțiunile sunt întotdeauna memorate în locații de memorie succesive. Ori de câte ori o instrucțiune este preluată din memorie în vederea executării ei, conținutul registrului PC este incrementat, astfel încât în fiecare moment el conține adresa următoarei instrucțiuni de executat. După executarea unei instrucțiuni de transfer de date sau de prelucrare de date, **controlul** revine la programul care dă succesiunea instrucțiunilor de executat.

Uneori, execuția programului de control poate schimba adresa conținută de PC şi astfel, să modifice succesiunea inițială de instrucțiuni de control de executat. Cu alte cuvinte, instrucțiunile de control programat precizează şi condițiile în care poate fi modificat conținutul registrului PC, dacă instrucțiunile de transfer sau de prelucrare solicită o astfel de modificare. Schimbarea valorii reținute în PC ca rezultat al executării unei instrucțiuni de control determină o pauză (*break*) în secvența de instrucțiuni de executat. Aceasta este o idee importantă pentru funcționarea sistemelor digitale deoarece determină controlarea fluxului de instrucțiuni, chiar cu posibilitatea de a executa bucle sau un segment de program auxiliar.

Cele mai importante instrucțiuni ale unui program de control sunt: **Branch, Jump, Skip, Call, Return, Compare** şi **Test.**

Instrucțiunile **Branch** şi **Jump** sunt similare şi determină încărcarea în PC a adreselor transmise ca parametri. In acest fel se asigură că următoarea instrucțiune se va executa de la adresa precizată, deci se poate realiza un salt în secvența de instrucțiuni predefinită. Aceste două instrucțiuni de control pot fi condiționate sau nu.

Instrucțiunea **Skip** determină salt peste instrucțiunea curentă, fără ca aceasta să fie executată. De asemenea, şi această instrucțiune poate fi condiționată sau nu.

Instrucțiunile **Call** şi **Return** se folosesc în contextul lucrului cu subrutine şi vor fi descrise ulterior.

Instrucțiunile **Compare** şi **Test** modifică indirect secvența instrucțiunilor de executat. Astfel, ele execută operații specifice de scădere şi respectiv SI logic. Rezultatul aplicării lor nu se reține, dar acesta modifică în mod corespunzător un anumit bit din registrul de stare.

Registrul de stare. Atunci când s-a convenit ca circuitul UAL să fie completat cu un registru de stare, biții acestuia vor fi utilizați în analiza

contextului curent, înainte sau după executarea anumitor instrucțiuni. Biții registrului de stare se numesc **biți de stare,** sau **biți de condiționare a codului** (*condition-code bits*), sau ***flag***-uri.

In figura următoare avem un registru de stare pe 4 biți, notați cu C (*Carry*), S (*Sign*), Z (*Zero*) şi V (*Overflow*). Aceşti biți sunt setați (puşi pe 1) sau resetați (puşi pe 0) în urma executării unei operații în UAL.

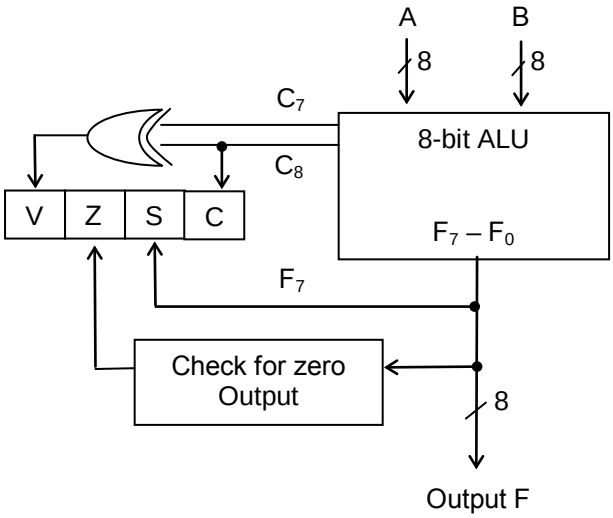
Semnificația celor patru biți de stare este următoarea:

C = 1 dacă bitul de transport C_8 este 1;

S = 1 dacă bitul de ordin maxim (bitul de semn) F_7 este 1;

Z = 1 dacă rezultatul furnizat de UAL este zero, adică conține numai zerouri;

V = 1 dacă s-a obținut o depăşire. Aceasta se interpretează diferit, în funcție de reprezentarea operanzilor în complement față de 2 sau față de 1.



Apelul şi revenirea din subrutine. O subrutină este o secvență de instrucțiuni care efectuează o acțiune de sine stătătoare, bine delimitată. In timpul executării unui program, o subrutină poate fi apelată de mai multe ori pentru a fi executată în diferite puncte ale programului principal. De fiecare dată când este apelată o subrutină, este executată automat o instrucțiune *branch* de deviere spre prima instrucțiune din subrutină pentru a inița executarea instrucțiunilor subrutinei. După executarea tuturor instrucțiunilor subrutinei, se execută o instrucțiune *branch* de revenire în programul principal.

Instrucțiunile care transferă controlul programului înspre şi dinspre subrutină pot fi: *call subroutine, jump to subroutine, branch to subroutine* sau *branch and save address*.

Apelul unei subrutine constă în două operații:

1. adresa următoarei instrucțiuni (cea reținută de PC) este memorată într-o locație temporară astfel încât subrutina să ştie unde să revină; aceasta va fi adresa de revenire – *return address*;
2. controlul este transferat la începutul subrutinei.

Ultima instrucțiune a oricărei subrutine, *return from subroutine*, realizează **revenirea în programul principal** prin transferarea adresei de revenire din locația temporară în PC.

Locația de memorie temporară pentru adresa de revenire diferă de la un SC la altul. Unele SC folosesc pentru aceasta prima locație de memorie a zonei rezervată subrutinei, altele folosesc o locație fixă din memorie, altele un registru de prelucrare, iar altele folosesc o stivă de memorie.

Varianta cea mai eficientă este cea cu stivă de memorie. In acest caz, fiecare apel de subrutină înseamnă o operație *push* pe stivă cu adresa de revenire din subrutina respectivă şi fiecare revenire înseamnă o operație *pop* de pe stivă pentru adresa respectivă. Avantajul apare la apelul în cascadă pentru mai multe subrutine deoarece organizarea memoriei ca stivă asigură că revenirea se face întotdeauna corect, adică în programul apelant (în ultimul program care a apelat procedura respectivă).

Astfel, o instrucțiune de apel constă în:

SP <— SP – 1

M[SP] <— PC

PC <— adresa efectivă a primei instrucțiuni a subrutinei

Dacă o altă subrutină este apelată din subrutina în execuție atunci o nouă adresă de revenire este depusă pe stivă, s.a.m.d.

O instrucțiune de revenire va consta în:

PC <— M[SP]

SP <— SP + 1

Folosind stiva pentru lucrul cu subrutine, programatorul nu mai trebuie să se concentreze asupra corectitudinii revenirii din subrutină în programul apelant.

Un al doilea avantaj al memorării adreselor de revenire pe stivă apare la lucrul cu subrutine recursive. Pe stivă nu interesează care adresă este depusă la un moment dat. Dacă sistemul ar lucra cu altă variantă de memorare a adreselor de revenire atunci ar trebui avut în vedere că este foarte posibil ca recursivitatea să se piardă prin distrugerea adresei de revenire memorate anterior pentru aceeaşi subrutină.

Intreruperea unui program. O **întrerupere** este o excepție sau un semnal intern sau extern transmis SC, prin care acestuia i se anunță apariția unui eveniment neobişnuit, adică ne-conform cu secvența de instrucțiuni în curs de execuție. Atunci când evenimentul s-a produs, au loc, în ordine, următoarele trei acțiuni:

1. suspendarea programului în curs de desfăşurare;
2. lansarea în execuție a unui program de depanare (*service program*), adică a unei rutine specializate, numită **Rutină de Tratare a Intreruperii (RTI)** sau **Handler de întrerupere,** care deserveşte întreruperea;
3. reluarea execuției programului suspendat.

Cauzele producerii acestor evenimente pot fi externe sau interne.

Exemple de evenimente de natură externă:

* apăsarea unei anumite combinații de taste, de exemplu CTRL/BREAK;

* terminarea unei operații de intrare/ieşire desfăşurată în *background* (adică în timp ce utilizatorul urmăreşte desfăşurarea altei acțiuni);

De cele mai multe ori, în urma tratării întreruperii externe, se reia activitatea programului suspendat.

Exemple de întreruperi de natură internă:

* împărțirea la zero;
* tentativa de adresare a unei zone de memorie care nu există;
* tentativa de execuție a unei instrucțiuni având un cod inexistent;
* depăşirea capacității de reprezentare a rezultatului unui calcul.

De cele mai multe ori, în urma tratării întreruperii interne, controlul este transmis SO, fără a mai relua activitatea programului suspendat.

Intreruperile externe sau interne sunt însoțite de semnale ale componentelor hardware ale SC. Față de acestea, în SC pot apărea şi evenimente inițiate de executarea unor instrucțiuni. Acestea vor fi *întreruperi software*. Corespunzător, avem rutine care deservesc apariția întreruperilor interne sau externe, care vor fi activate la apariția evenimentului, fără a se putea prevedea momentul apariției. Vom spune despre *handler*-ele din această categorie că *sunt activate prin evenimente.* În schimb, cealaltă parte a *handler*-elor oferă aceleaşi servicii ca o bibliotecă de subprograme, apelabile de către programele utilizator. Activarea lor are loc atunci când sunt solicitate prin program. *Handler*-ele din această categorie *sunt activate prin instrucțiuni speciale de apel de întreruperi.*

La apariția unei întreruperi SC trebuie, în ordine:

1) să determine tipul evenimentului care a generat întreruperea (intern, extern);
2) să afle care este sursa (cauza) întreruperii;
3) să determine adresa RTI (a rutinei de tratare a întreruperii), adică în ce manieră se poate identifica *handler-ul* corespunzător.

Rezolvarea acestor probleme presupune că pentru fiecare tip de eveniment şi pentru fiecare cauză posibilă a unei întreruperi să existe câte un *handler* specific. *Metoda cea mai folosită* pentru localizarea rapidă a RTI este **vectorizarea întreruperilor**. Aceasta constă în a asocia, pentru fiecare întrerupere, o locație de memorie cu adresă fixă. În această locație fixă se trece adresa RTI corespunzătoare întreruperii. Deci, în momentul apariției unei înteruperi, mai întâi se preia din locația de memorie asociată întreruperii adresa handlerului corespunzător ei şi apoi se cedează controlul de execuție *handler*-ului.

O parte dintre rutinele de tratare a întreruperilor sunt furnizate o dată cu sistemul de calcul. O altă parte sunt scrise de către proiectanții sistemului de operare. În sfârsit, utilizatorii pot să-şi scrie propriile lor RTI.

În proiectarea unui handler de întrerupere trebuie respectate, în ordine, următoarele etape:

1) Salvarea contextului programului suspendat: regiştri, flag-uri, stiva etc. De obicei, SC reține contextul curent prin memorarea conținutului registrului PC şi a cuvântului PSW – *program status word*. Acesta caracterizează UCP deoarece coține valorile curente ale biților de stare.

2) Interzicerea tratării întreruperilor care pot perturba evoluția normală a handler-ului respectiv. De exemplu, handler-ul care rezolvă împărțirea la zero poate să conțină, eventual din greşeală, o împărțire la zero. În acest caz ar trebui să nu fie apelat din nou acelaşi handler.

3) Rezolvarea situației care a generat întreruperea, aceasta constituind partea esențială a handler-ului.

4) Ridicarea restricțiilor impuse la etapa 2.

5) Refacerea contextului salvat la etapa 1.

6) Predarea controlului către programul care a fost suspendat sau către sistemul de operare.

Regula unanim acceptată de către proiectanții sistemelor de întreruperi este aceea că orice întrerupere este luată în considerare numai între execuțiile a două instrucțiuni maşină succesive. Niciodată UCP nu răspunde unei întreruperi până când nu a încheiat execuția instrucțiunii curente. Inainte de fiecare fază de preluare a următoarei instrucțiuni din memorie, unitatea de control verifică şi rezolvă toate semnalele de întrerupere. Dacă apar simultan două întreruperi, circuitele hard ale SC trebuie să decidă care dintre ele va fi deservită prima.

Rezumat.

In acest paragraf sunt prezentate principalele instrucțiuni recunoscute de procesor într-un program de control. Aceastea sunt grupate în funcție de rolul lor în sistem:

(1) instrucțiuni pentru transferul datelor

a) transfer între memorie şi regiştri de prelucrare

b) transfer între regiştri de prelucrare şi stivă

c) transfer între doi regiştri de prelucrare

(2) instrucțiuni pentru gestionarea datelor

a) instrucțiuni aritmetice

b) instrucțiuni logice şi de prelucrare la nivel de bit

c) instrucțiuni de deplasare

(3) instrucțiuni de control al programului.

Cuvinte cheie.

instrucțiune
registru de stare
subrutină
întrerupere, vectorizarea întreruperilor, tratarea întreruperilor

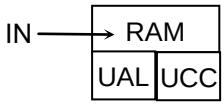
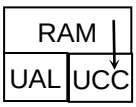
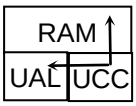
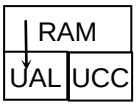
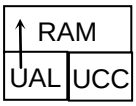
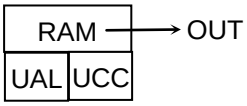
Verificare.

1. Ce este o subrutină? De ce este potrivită utilizarea stivei pentru gestionarea apelului şi revenirii dintr-o subrutină?
2. Ce este o întrerupere? Ce se înțelege prin *vectorizarea întreruperilor*?

NOTIȚE

3.7. Prelucrarea instrucțiunilor unui program

Prelucrarea instrucțiunilor unui program de către UCP constă în principal în următoarele etape:

1.	UCC emite semnale de comandă către dispozitivul periferic de intrare pentru transferul de instrucțiuni şi de date către memoria RAM;	
2.	UCC extrage instrucțiunile din memoria internă şi le interpretează (decodifică);	
3.	în funcție de interpretarea făcută, UCC emite semnale de comandă către UAL şi RAM;	
4.	datele auxiliare solicitate din RAM sunt transferate în UAL, unde se execută, sub supravegherea UCC, operația aritmetică sau logică decodificată;	
5.	rezultatele obținute în UAL sunt dirijate de UCC către RAM;	
6.	se repetă etapele de la 1. la 5. pentru fiecare instrucțiune;	
7.	După executarea tuturor instrucțiunilor, UCC direcționează RAM să transfere rezultatele către un dispozitiv periferic de ieşire.	

Verificare.

1. Identificați în structura fizică a sistemului de calcul unitățile funcționale corespunzătoare componentelor implicate în acest paragraf.

NOTIȚE

3.8. Microprocesorul

Indiferent de structura sa internă, UCP este cea mai importantă componentă pentru funcționarea unui SC. De aceea este numită şi *creierul sistemului de calcul*. Deoarece UCP se concretizează fizic în componente numite **procesoare**, rezultă de aici că procesorul (în diversele sale variante) este unitatea de prelucrare a sistemului, deci componenta fundamentală a unui sistem de calcul.

La supercalculatoare şi la calculatoarele medii-mari, UCP poate fi construită prin circuite structurate pe un cadru principal, numit **mainframe**, în timp ce, la microcalculatoare, UCP este construită din circuite integrate structurate pe pastile de siliciu care vor forma **microprocesoarele.**

3.8.1. Caracteristici

Puterea unui microcalculator este dată de:

- ✓ viteza de executie,
- ✓ memorie şi
- ✓ acuratețea circuitelor electronice componente.

Un microprocesor se apreciază în funcție de:

- ✓ viteza cu care execută instrucțiunile şi
- ✓ cantitatea de memorie internă pe care o poate accesa în mod direct.

Viteza de execuție a unui microprocesor depinde, la rândul ei, de:

- ✓ lungimea cuvântului şi
- ✓ viteza ceasului.

Lungimea cuvântului reprezintă capacitatea registrului de bază, adică numărul de biți ce poate fi prelucrat la un moment dat de microprocesor. Această valoare este invers proporțională cu viteza de execuție. Primele microprocesoare pe 8 biți au fost create de firma ZILOG şi numite Z80. Cronologic, urmează microprocesoarele din familia Intel care au lungimea cuvântului de 16 biți (I 8086, I 8088, I 80286), de 32 biți (I 80386, I 80486) sau de 64 biți pentru microprocesoarele Pentium.

Viteza ceasului sau ***frecvența de tact*** reprezintă numărul de impulsuri electronice pe care le poate produce ceasul intern al microprocesorului într-o secundă. Viteza ceasului este măsurată în Hz şi multiplii „informatici" ai acestora. Actualmente, microprocesoarele produse au depăşit viteza ceasului de 1,5 GHz.

Memoria internă pe care o poate accesa în mod direct depinde de lungimea cuvântului. Astfel, pentru cuvânt de 16 biți microprocesorul va putea accesa 2^{16} octeți, în timp ce pentru un cuvânt de 32 biți, microprocesorul va putea accesa 2^{32} octeți. In tabelul următor prezentăm pe scurt evoluția cronologică a microprocesoarelor, precum şi principalele caracteristici:

μ-procesor	Anul	Lg cuvânt	-MHz-	-Mips-
I 8086	1978	16	5-10	0,33

I 8088	1981	16	4	0,33
I 80286	1982	16	6	1,2
I 80386 SX	1985	32	20	2,5
I 80386 DX		32	20 şi 30	6
I 80486 SX	1989	32	33	16,5
I 80486 DX		32	> 33	20
I 80486 DX/2		32	50 şi 66	54
I 80486 DX/4		32	100	75
Pentium	1993	64	66	112

Mips (*Mega-instrucțiuni pe secundă*) este o caracteristică ce măsoară capacitatea de prelucrare.

Deoarece este componenta fundamentală a SC, microprocesorul este cel care determină şi denumirea calculatorului gazdă. Prezentăm în continuare microprocesoarele din familia Intel I 80x86 şi sistemele care le includ.

Microprocesor	**Calculator personal**
I 8086	IBM PC
I 8088	IBM PC XT
I 80286	IBM PC AT 286
I 80386 SX	IBM PC AT 386 SX
I 80386 DX	IBM PC AT 386 DX
I 80486 SX	IBM PC AT 486 SX
I 80486 DX	IBM PC AT 486 DX
I 80486 DX/2	IBM PC AT 486 DX2
I 80486 DX/4	IBM PC AT 486 DX4

IBM - **I**nternational **B**usiness **M**achine (“*Big Blue*”)

XT - e**XT**ended

AT - **A**dvanced **T**echnology

La apariția sa în 1978, procesorul 8086 era cu adevărat o unitate de prelucrare cu un singur cip pe 16 biți. Proiectul lui 8086 era oarecum similar cu cel al lui 8080, dar nu integral compatibil. Cipul 8086 a fost urmat de cipul 8088, care avea aceeaşi arhitectură ca şi 8086 şi rula aceleaşi programe, dar avea o magistrală pe 8 biți şi nu pe 16 biți, fiind deci mai lant, dar mai ieftin decât 8086. acest cip a devenit rapid standard pentru industria calculatoarelor personale odată cu alegerea sa de către IBM pentru producerea sistemului inițial IBM PC.

Nici 8088, nici 8086 nu puteau adresa mai mult de 1MO de memorie. Astfel, la începutul anilor ’80, Intel a proiectat 80286, o versiune îmbunătățită , compatibilă cu 8086. următorul pas a fost un adevărat procesor pe 32 biți şi anume 80386, compatibil cu procesoarele anterioare. Această compatibilitate era oun plus pentru cei care doreau să ruleze în continuare vechile soft-uri şi un inconvenient pentru cei care ar fi preferat o arhitectură simplă, modernă, curată, fără impedimentele greşelilor şi tehnologiile trecutului.

Incercând să răspundă cerințelor pieții, în 1989 Intel scoate procesorul 80486: o versiune mai rapidă a lui 80386, cu coprocesorul matematic inclus şi cu o memorie cache de 8KO inclusă pe cip. Procesorul 486 mai avea incorporat şi suportul pentru multiprocesare, pentru a permite producătorilor de calculatoare să construiască sisteme cu mai multe UCP.

Pentru generația următoare de procesoare, Intel a avut surpriza să descopere că, de fapt, numerele (ca 80486) nu pot fi mărci înregistrate. Astfelcă, următoarea generație de procesoare se va numi Pentium, de la grecescul $\pi\varepsilon\nu\delta\varepsilon$ (cinci). Spre deosebire de 80486 care avea o singură bandă de asamblare internă (*pipeline*), Pentium are două astfel de benzi, deci este de ori mai rapid. Următoarea generație de procesoare este Pentium Pro, o rupere clară față de trecut pentru că noul Pentium Pro avea o organizare internă total diferită şi putea să execute până la cinci instrucțiuni simultan. In plus, memoria cache internă avea două nivele: 8KB pentru instrucțiuni şi 8KB pentru date pe un nivel şi încă 256KB pe al doilea nivel.

In continuarea seriei, noul procesor Intel a fost Pentium II, în fapt un Pentium Pro cu extensii speciale pentru multimedia (MMX, *MultiMedia eXtension*). Aceste instrucțiuni aveau rolul de a accelera calculele necesare pentru procesarea semnalelor audio şi video, înlăturând astfel nevoia unor coprocesoare speciale multimedia.

La începutul anului 1998, Intel a introdus o linie nouă de produse numită Celeron – o variantă a procesorului Pentium II ieftină, cu performanți scăzute, destinată PC-urilor de duzină.

3.8.2. Setul de instrucțiuni ale microprocesorului

Un aspect important al arhitecturii unui SC este dat de setul de instrucțiuni recunoscute de microprocesorul acelui sistem. De aceste instrucțiuni depinde construcția şi folosirea limbajului maşină al calculatorului în discuție.

La început, datorită posibilităților reduse ale echipamentelor hardware, calculatoarele lucrau cu instrucțiuni puține şi simple. Ulterior, apariția circuitelor integrate a permis mărirea numărului de instrucțiuni recunoscute de un microprocesor, ajungându-se la sisteme cu 100 sau chiar 200 de instrucțiuni în setul de bază. Implicit, aceste SC pot prelucra un număr mare de tipuri de date şi diverse moduri de adresare a memoriei.

In 1980, un grup de la Berkeley, condus de David Patterson şi Carlo Séquin, a îmceput să proiecteze cipuri VLSI pentru UCP fără interpretor. Ei au introdus termenul RISC pentru acest concept şi şi-au numit cipul de unitate centrală RISC I, urmat la scurt timp de RISC II. Puțin mai târziu, în 1981, la Stanford, John Hennessy a proiectat şi fabricat un cip întrucâtva diferit, pe

care l-a numit MIPS. Aceste cipuri au evoluat în produse comerciale importante, procesoarele SPARC şi respectiv MIPS. De fapt, acronimul MIPS este pentru *Milioane de Instrucțiuni pe Secundă*, procesorul MIPS fiind denumit astfel numai printr-un joc de cuvinte. Ulterior, structura internă a acestui procesor a permis şi o denumire potrivită: *Microprocessor without Interlocked Pipeline Stages*.

Majoritatea microprocesoarelor folosite în prezent fac parte din familia **CISC** (**C**omplex **I**nstruction **S**et **C**omputer). Cele mai performante microprocesoare de acest tip sunt **Pentium**-urile firmei Intel şi **M68040** al firmei Motorola, ultimul echipând PC-urile de tip MacIntosh.

Tendința actuală se îndreaptă spre microprocesoarele de tip **RISC** (**R**educed **I**nstruction **S**et **C**omputer). In general, acestea lucrează pe 64 biți, cu o viteză de până la 200 MHz. La baza arhitecturii RISC stă intenția unor constructori de microprocesoare de a reduce timpul de execuție a unui program prin simplificarea setului de instrucțiuni. Aceştia au recomandat ca SC să folosească mai puține instrucțiuni, care să aibă structuri simple, astfel încât să poată fi executate mai rapid în UCP, adesea fără un necesar de memorie suplimentară.

Procesoarele MIPS apar ca un caz particular de procesoare RISC şi sunt folosite astăzi în special pe stațiile de lucru Silicon Graphics.

Pe scurt, caracterizarea celor două arhitecturi ar fi:

CISC	1. număr mare de instrucțiuni (100 – 250); 2. unele instrucțiuni execută acțiuni specializate (se apelează rar); 3. număr mare de moduri de adresare a memoriei (5 – 20); 4. formatul instrucțiunilor nu are lungime fixă; 5. există instrucțiuni care prelucrează operanzi direct în memorie;
RISC	1. relativ puține instrucțiuni; 2. relativ puține moduri de adresare a memoriei; 3. există acces limitat la memorie pentru instrucțiunile *Load* şi *Store*; 4. toate operațiile se efectuează în regiştrii UCP; 5. instrucțiunile au format fix (sunt uşor de decodificat); 6. control bazat pe circuite (preferabil controlului microprogramat).

3.8.3. Modelarea funcționării unui procesor

In acest paragraf vom prezenta principiile generale de funcționare ale unui procesor, considerând două modele, alese pe considerente didactice să reprezinte simplu şi pe înțeles procesorul. Această modalitate de prezentare a rezultat în urma unui proces de simplificare a construcției unui procesor. De aceea, în practică nu se pot regăsi aceste modele foarte simple, dar ele constituie scheletul oricărui model de procesor.

3.8.3.1. Maşina cu trei adrese

Modelul maşinii cu trei adrese conține în memorie atât **datele** cu care lucrează, cât şi **instrucțiunile** care descriu programul de executat.

Zonele de date le vom identifica prin numele lor. *Presupunem că fiecare dată încape pe un cuvânt*.

Formatul instrucțiunilor.

O instrucțiune la maşina cu trei adrese ocupă patru cuvinte:

cod	adr1	adr2	adr3

Semnificația cuvintelor este:

- ✓ cod – indică codul operației de efectuat;
- ✓ adr1, adr2, adr3 – indică trei locații din memoria maşinii.

O astfel de instrucțiune se execută prin:

adr3 := (adr1) cod (adr2)

adică, între numărul conținut în locația de memorie indicată de adr1 (ca prim operand) şi numărul conținut în locația indicată de adr2 (ca al doilea operand) se efectuează operația indicată de cod. Numărul obținut ca rezultat este depus în locația indicată de adr3.

Instrucțiunile se execută în secvență, una după alta. Schimbarea ordinii de execuție a instrucțiunilor se face cu ajutorul instrucțiunilor de salt.

Pentru a ne referi mai uşor la ele, unele instrucțiuni pot fi identificate printr-un nume:

nume:	cod	adr1	adr2	adr3

Instrucțiunile maşinii cu trei adrese.

Maşina cu trei adrese lucrează cu trei tipuri de instrucțiuni:

1. instrucțiuni de calcul;
2. instrucțiuni de salt;
3. instrucțiuni speciale.

In continuare notăm adresele de memorie cu litere mari (A, B, C etc.), atât pentru locațiile din zona de date cât şi pentru cele din zona de instrucțiuni.

1. Maşina cu trei adrese recunoaşte 7 **instrucțiuni de calcul** şi anume, patru instrucțiuni pentru operațiile aritmetice fundamentale şi trei instrucțiuni pentru operații unare uzuale:

+	A	B	C	Adunare	C:=A+B
–	A	B	C	Scădere	C:=A–B
*	A	B	C	Inmulțire	C:=A*B

/	A	B	C	Impărțire	C:=A/B
√	A	-	C	Radical de ord 2	C:= √A
| |	A	-	C	Valoare absolută	C:=|A|
[ ]	A	-	C	Partea întreagă	C:=[A]

2. Instrucțiunile de salt sunt de două tipuri: de salt condiționat şi de salt necondiționat.

Maşina cu trei adrese recunoaşte o singură *instrucțiune de salt necondiționat*:

SN	-	-	A	Următoarea instrucțiune care se va executa este cea de la adresa A

O *instrucțiune de salt condiționat* comandă saltul la instrucțiunea de la adresa adr3 numai dacă numerele de la adresele adr1 şi adr2 se află în relația de ordine specifică instrucțiunii respective. Maşina cu trei adrese recunoaşte 6 instrucțiuni de salt condiționat:

<	A	B	C	Salt la instrțiunea indicată prin C dacă A < B
≤	A	B	C	Salt la instrțiunea indicată prin C dacă A ≤ B
>	A	B	C	Salt la instrțiunea indicată prin C dacă A > B
≥	A	B	C	Salt la instrțiunea indicată prin C dacă A ≥ B
=	A	B	C	Salt la instrțiunea indicată prin C dacă A = B
≠	A	B	C	Salt la instrțiunea indicată prin C dacă A ≠ B

3. Instrucțiunile speciale ale maşinii cu trei adrese sunt: CIT, TIP şi STOP.

CIT	-	-	A	Citeşte de la intrare valoarea unui număr şi o depune în locația indicată prin adresa A
TIP	-	-	A	Afişează la ieşire valoarea conținută în locația indicată prin adresa A
STOP	-	-	-	Incheie execuția programului (oprire)

3.8.3.2. Maşina cu o adresă

Arhitectura maşinii cu o adresă este ceva mai complexă decât a modelului prezentat anterior, astfel că se apropie mai mult de construcția reală a unui procesor.

Structura memoriei la maşina cu o adresă.

Principalele componente ale memoriei maşinii cu o adresă sunt:

1. *regiştrii generali*;
2. zona de memorie rezervată pentru *program*;
3. zona de memorie rezervată pentru *date*;
4. zona de memorie rezervată pentru *stivă*.

1. Regiştrii maşinii cu o adresă sunt trei:

1.	registrul acumulator, **A**	– are capacitatea de un cuvânt; – reține la un moment dat unul dintre operanzii operației ce urmează a fi executată şi după execuție reține rezultatul operației;
2.	registrul index, **I**	– are capacitatea de un cuvânt; – este folosit pentru a accesa componentele unui vector de numere memorat în zona de date;
3.	registrul de bază, **B**	– este utilizat pentru a se permite comunicarea între diferite părți ale unui program.

Observație. Depunerea unui număr într-un registru face ca vechiul număr din registru să se piardă. In acelaşi timp, conținutul unui registru nu se pierde până când nu se depune un nou număr.

2. Zona de program conține instrucțiunile care urmează a fi executate.

O instrucțiune se reprezintă pe două cuvinte şi are formatul:

nume:	cod	adr

Semnificația câmpurilor din format este dată de:

A := (A) cod (adr)

adică, se execută operația indicată de cod folosind conținutul curent al registrului acumulator, A şi conținutul locației de la adresa adr.

3. Zona de date este zona în care sunt memorate valorile variabilelor cu care lucrează programul. Fiecare variabilă simplă se reprezintă pe un cuvânt. Pe lângă variabilele simple, la această maşină se poate opera uşor şi

cu variabile cu indici: vectori, matrice bidimensionale, tablouri tridimensionale etc. Acestea din urmă vor fi memorate element cu element în locații succesive.

Observație. Depunerea unui număr într-o locație face ca vechiul număr memorat să se piardă. In acelaşi timp, conținutul nu se pierde până când nu se depune un nou număr.

4. Stiva este o zonă de memorie pentru care accesul la locații este gestionat de disciplina LIFO şi de următoarele reguli:

– în fiecare moment se depune sau se extrage un singur cuvânt din stivă;

– nu se poate extrage un număr din stivă decât dacă înainte au fost deja depuse numere în stivă;

– la extragerea unui număr din stivă numărul respectiv se şterge de pe stivă;

– numerele neextrase din stivă îşi păstrează valorile.

Observație. Tentativele de efectuare a unei operații de depunere într-o stivă plină sau de extragere dintr-o stivă vidă (situațiile de *depăşire*) conduc la oprirea maşinii cu o adresă.

Instrucțiunile maşinii cu o adresă.

Pentru prezentarea instrucțiunilor maşinii cu o adresă convenim să notăm cu **X** o adresă oarecare din zona de date, sau o adresă din zona de program, sau o constantă.

Maşina cu o adresă recunoaşte 28 instrucțiuni considerate în patru grupe:

a) instrucțiuni pentru transferuri de date;
b) instrucțiuni de calcul;
c) instrucțiuni de salt;
d) instrucțiuni speciale.

Pentru maşina cu o adresă putem identifica o instrucțiune în mod unic şi printr-un număr de la 1 la 28.

In tabelele următoare, pentru fiecare instrucțiune precizăm: numărul de ordine, codul, adresa locației pe care se execută instrucțiunea şi, eventual, comentarii specifice fiecărei instrucțiuni.

a) **Instrucțiunile pentru transferuri de date** sunt:

1	$\leftarrow$	X	Valoarea indicată de X se depune în registrul acumulator A
2	$\rightarrow$	X	Valoarea din registrul acumulator A se depune în locația din zona de date indicată de X
3	$\leftarrow_I$		Conținutul registrului index I se depune în registrul acumulator A
4	$\rightarrow_I$		Conținutul registrului acumulator A se depune în registrul index I

5	$\leftarrow_B$		Conținutul registrului de bază B se depune în registrul acumulator A
6	$\rightarrow_B$		Conținutul registrului acumulator A se depune în registrul de bază B
7	↓	X	Valoarea indicată prin X este depusă în vârful stivei; numărul elementelor din stivă creşte cu 1
8	↑	X	Este extras elementul din vârful stivei şi este depus în locația de date indicată de X; numărul elementelor din stivă scade cu 1
9	↓		Valoarea din registrul acumulator A este depusă în vârful stivei; numărul elementelor din stivă creşte cu 1
10	↑		Este extras elementul din vârful stivei şi este depus în registrul acumulator A; numărul elementelor din stivă scade cu 1

b) **Instrucțiunile de calcul** coincid cu cele recunoscute de maşina cu trei adrese:

11	+	X	Adunare	A:=A+X
12	–	X	Scădere	A:=A–X
13	*	X	Inmulțire	A:=A*X
14	/	X	Impărțire	A:=A/X
15	√	X	Radical de ord 2	A:= √X
16	| |	X	Valoare absolută	A:=|X|
17	[ ]	X	Partea întreagă	A:=[X]

c) **Instrucțiunile de salt** pot fi condiționate sau nu. Si maşina cu o adresă recunoaşte o singură *instrucțiune de salt necondiționat*:

18	SN	X	Următoarea instrucțiune care se va executa este cea de la adresa din zona de program indicată de X

Instrucțiunile de salt condiționat determină un salt la instrucțiunea indicată de X <u>dacă</u> conținutul acumulatorului A este într-o anumită relație de ordine față de zero. Avem următoarele 6 instrucțiuni de salt condiționat:

19	<	X	Salt la X dacă A < 0

20	$\leq$	X	Salt la X dacă A $\leq$ 0
21	>	X	Salt la X dacă A > 0
22	$\geq$	X	Salt la X dacă A $\geq$ 0
23	=	X	Salt la X dacă A = 0
24	$\neq$	X	Salt la X dacă A $\neq$ 0

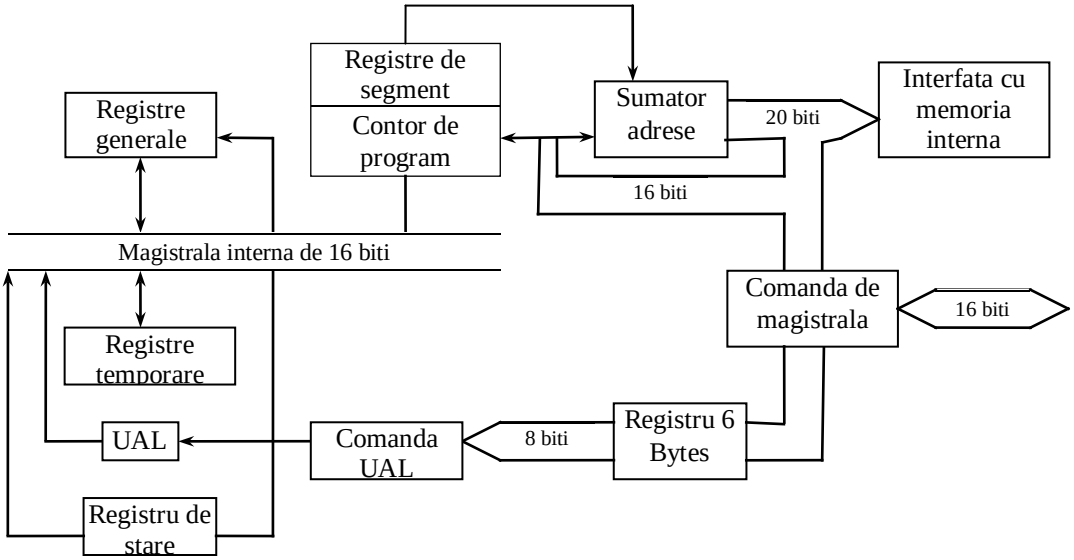
d) **Instrucțiunile speciale** ale maşinii cu o adresă completează cu o instrucțiune setul de la maşina cu trei adrese:

25	ADR	X	Determină adresa (nu valoarea) din zona de date a variabilei indicate de X şi depune această adresă în registrul acumulator A
26	CIT	X	Citeşte de la intrare valoarea unui număr şi o depune în locația din zona de date indicată de X
27	TIP	X	Afişează la ieşire valoarea numărului aflat în locația din zona de date indicată de X
28	STOP		Incheie execuția programului (oprire)

NOTIȚE

3.8.4. Structura internă de bază a unui microprocesor I 80x86

Complicând modelele din paragraful anterior, prezentăm aici schema unui microprocesor din familia Intel 80x86 pe 16 biți.



Registrele generale sunt:

a) ***registrele de utilizare generală*** se notează cu:

- AX - registrul acumulator general
- BX - registrul de bază
- CX - registrul numărător
- DX - registrul de date.

Acestea pot fi adresate în două moduri: direct pe 16 biți sau adresând separat byte-ul superior (primii 8 biți) şi byte-ul inferior (următorii 8 biți). Corespuzător, cuvintele se vor numi:

15 8	7 0
AH	AL
BH	BL
CH	CL
DH	DL

b) ***registrul indicator de stivă*** de 16 biți. Acesta conține adresa vârfului stivei, SP (*stack pointer*).

c) ***registrul indicator de bază*** de 16 biți. Acesta conține adresa de bază, BP (*base pointer*).

d) ***registrele de index***: SI (*source index*) şi DI (*destination index*), de 16 biți fiecare. Aceştia participă la elaborarea adreselor, fiecare adresă rezultând prin însumarea diferitelor combinații între adresa de bază, un indice relativ la poziția curentă şi o deplasare (numita *adresă de offset*).

Registrul de stare este registrul indicatorilor de condiție. Fiecare subrutină care foloseşte registrul de flaguri va interpreta de o manieră proprie valorile găsite. Cel mai des, indicatorii de condiție sunt folosiți în instrucțiunile de salt.

Cei 16 indicatori de condiție ai registrului de flag-uri au următoarea semnificație:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	**X**	**X**	**X**	**OF**	**DF**	**IF**	**TF**	**SF**	**ZF**	**X**	**AF**	**X**	**PF**	**X**	**CF**

X - rezervați;

CF - indicator de transport din MSB al rezultatului;

PF - indicator de paritate;

AF - indicator de transport de la bitul 4 la bitul 5;

ZF - indicator de rezultat nul;

SF - indicator de semn negativ;

TF - indicator de întrerupere internă;

IF - indicator de întrerupere externă permisă;

DF - indicator de decrementare / incrementare a regiştrilor SI sau DI după execuția unei operații;

OF - indicator de depăşire aritmetică.

Registrele de segment ale microprocesorului permit adresarea memoriei interne, fiecare având o capacitate de 16 biți. Aceştia sunt:

CS - Cod Segment - segmentul pentru program (cod sursă);

DS - Data Segment - segmentul pentru date;

SS - Stack Segment - segmentul pentru stivă;

ES - Extra Segment - segmentul pentru date auxiliare.

Contorul de program, PC sau **IP** (*Instruction Pointer*) are o capacitate de 16 biți şi conține o adresă relativă (*offset*) a instrucțiunii curente din segmentul de program CS.

De exemplu, pentru o instrucțiune de salt, IP este salvat în vârful stivei şi încărcat cu adresa relativă în segmentul program.

Registrul de 6 bytes conține instrucțiuni şi are rolul de a îmbunătăți viteza de prelucrare prin două mecanisme distincte:

1) datele preluate din memorie sunt plasate în acest registru în ordinea introducerii; astfel, UAL poate să preia instrucțiunile fără să fie influențată de timpul de acces al memoriei;

2) la transferul operanzilor între memorie sau dispozitive periferice şi UAL se generează şi adresele necesare.

Rezumat.

In prezentarea structurii organizatorice şi funcționale a unui sistem de calcul un loc aparte trebuie rezervat microprocesorului – creierul calculatorului şi reprezentarea fizică cea mai importantă a UCP.

In prima parte a acestui paragraf este descris procesorul din diferite puncte de vedere: istoric, caracteristici generale, caracteristici de performanță. Apoi sunt tratate aspecte concrete legate de cele trei tipuri de arhitecturi de procesoare existente pe piață: CISC, RISC şi MIPS.

In pregătirea capitolului al patrulea, paragrafele 3.8.3 şi 3.8.4 prezintă trei *nivele* de structuri de unități de prelucrare microprogramate. Este vorba de modelul maşinii cu trei adrese, modelul maşinii cu o adresă şi, în final, modelul larg răspândit al procesoarelor din familia Intel. Aceste trei exemple reprezintă totodată trei nivele de abstractizare (de la simplu la complex) a structurii interne a unității de prelucrare. Cel mai simplu model este cel al maşinii cu trei adrese pentru că toate instrucțiunile sunt în mod de adresare directă. La maşina cu o adresă, deja modelul este mai complicat pentru că multe dintre instrucțiunile recunoscute de această maşină lucrează cu regiştrii sau cu alte zone de memorie implicite.

Cuvinte cheie.

procesor
lungimea cuvântului, frecvența de tact
instrucțiune, cod de instrucțiune, setul de bază de instrucțiuni
arhitectură CISC, RISC, MIPS

Verificare.

1. Dacă un procesor are frecvența de tact de 6MHz aceasta înseamnă că poate executa 6 milioane de operații într-o secundă. Dacă considerăm că o astfel de operație este o etapă din *pipeline*-ul intern al procesorului respectiv, să se calculeze viteza de prelucrare a acestui procesor (în Mips). Se va considera că banda de asamblare este cu cinci faze.

3.9. Coprocesoare matematice

Pentru a spori viteza de execuție a operațiilor matematice, arhitectura internă a microprocesoarelor s-a extins cu o unitate specializată în operații cu numere reale exprimate în virgulă mobilă, numită **coprocesor matematic.** Pentru utilizator, această unitate auxiliară este integrată în microprocesorul părinte prin interfața coprocesorului. Cele două componente sunt conectate în paralel, astfel că fiecare poate să-şi execute la un moment dat propriile sale instrucțiuni.

Coprocesoarele matematice au urmat evoluția microprocesoarelor. Până la microprocesorul I80486SX coprocesorul este un *chip* distinct de microprocesor şi preia numele acestuia, schimbând doar ultima cifră. Astfel, coprocesorul matematic ataşat microprocesorului I80x86 se va numi I 80x87. După varianta I80486SX, microprocesoarele I80486DX, I80486DX2, I80486DX4 şi Pentium au inclus coprocesorul matematic pe acelaşi *chip* cu microprocesorul.

In absența coprocesoarelor, programele folosesc *emulatoare de coprocesoare* care sunt conectate în locul microprocesoarelor preluând astfel controlul întregului sistem.

3.10. Multiprocesare

Dacă două sau mai multe calculatoare sau procesoare cooperează într-o manieră oarecare atunci totalitatea resurselor lor formează un **sistem distribuit**. De aici rezultă imediat două tipuri de sisteme distribuite: sistemele **multicalculator** şi sistemele **multiprocesor**.

Există numeroase asemănări între sistemele multiprocesor şi sistemele multicalculator. Cele mai multe asemănări rezultă din faptul că ambele tipuri suportă operații efectuate paralel şi / sau concurent.

Totuşi, există o deosebire importantă între un sistem cu mai multe procesoare şi un sistem cu mai multe calculatoare. Intr-un sistem multicalculator, calculatoarele sunt interconectate între ele prin linii de comunicare formând o **rețea de calculatoare**. Intr-o rețea avem mai multe calculatoare autonome (independente) care pot sau nu să comunice fiecare cu fiecare. Comparativ, un sistem multiprocesor este controlat de un acelaşi sistem de operare, care asigură interconexiunea între procesoare şi toate componentele care concură la realizarea sarcinilor.

Cu alte cuvinte, un sistem multiprocesor rezultă din interconectarea a două sau mai multor procesoare cu memorii şi echipament de intrare / ieşire.

Din această definiție trebuie să înțelegem că un procesor poate reprezenta o unitate UCP sau poate fi un procesor IOP. Totuşi, calitatea de sistem multiprocesor apare când cel puțin unul dintre procesoare are proprietăți de UCP şi nu numai de comunicare de tip IOP.

Sistemele multiprocesor fac parte din categoria MIMD.

Rețelele locale sunt apreciate ca cele mai adecvate şi răspândite suporturi fizice pentru sistemele cu prelucrare distribuită.

Multiprocesarea creşte fiabilitatea sistemului prin faptul că o eroare apărută într-o *parte* are un efect limitat şi nu influențează întregul sistem. Mai mult, dacă un procesor *cade*, procesele lui pot fi dirijate spre execuție către un alt procesor.

Un avantaj fundamental al sistemelor multiprocesor constă în posibilitatea procesării paralele a task-urilor în două moduri:

1. mai multe sarcini independente pot fi executate în paralel sau

2. o aceeaşi sarcină poate fi împărțită în mai multe task-uri de executat în paralel.

Rolul acestui paragraf este de a deschide noi orizonturi pentru studentul care şi-a însuşit noțiunile elementare de arhitectura calculatoarelor prezentate în acest curs şi care este astfel în măsură să înțeleagă structuri mai complexe de organizare şi realizare a conexiunilor inter-calculatoare.

Multiprogramarea reprezintă modul de exploatare a unui sistem de calcul cu un singur procesor central, dar care presupune existența simultană în memoria internă a mai multor programe care se execută concurent (procese concurente); astfel rezultă o mai bună utilizare a UCP şi a memoriei.

Un **task** este unitatea elementară delimitată într-o secvență de acțiuni specifice pentru atingerea unui anumit scop. Termenul de **multitasking** sau **multiprelucrare** se referă la capacitatea unui calculator de a executa mai multe taskuri simultan. La o prezentare mai detaliată a conceptului, caracteristica de multitasking se atribuie direct sistemului de operare. Astfel, politica de alocare a taskurilor la unitățile de prelucrare disponibile în sistem ține de organizarea şi gestionarea resurselor de către sistemul de operare gazdă. Un task în execuție este un **proces**. Dacă, la un moment dat, sunt executate mai multe procese simultan atunci aceste sunt **procese paralele**. Dacă, în plus, aceste procese evoluează cu interschimb de informații între ele, atunci ele sunt **procese concurente**. Din alt punct de vedere, dacă sistemul gazdă dispune de mai multe unități de prelucrare pentru a executa procesele paralele atunci **paralelismul este efectiv**. Altfel, **paralelismul este simulat** prin diferite politici de alocare proces-procesor.

In ceea ce priveşte aspectele de paralelism la nivelul structurii interne a sistemului de calcul, se poate spune că în arhitectura calculatorelor Mark I şi ENIAC intrau în calcul la un moment dat mai multe elemente de calcul ce lucrau în paralel la o problemă comună, fiind dirijate de o singură unitate de comandă. La versiunile următoare s-a renunțat la această arhitectură paralelă. Ulterior, în arhitectura procesoarelor Intel s-a revenit la paralelizarea etapelor de prelucrare. Astfel, dacă la 8086 erau doar două unități funcționale care lucrau în paralel (EU şi BIU), structura procesorului 286 conține 4 unități funcționale paralele, în timp ce procesorul 386 conține 6 astfel de unități funcționale şi anume:

1. unitatea de interfață cu magistrala (BIU, engl. *Bus Interface Unit*);
2. unitatea de citire în avans a instrucțiunilor (engl. *Code Prefetch Unit*);
3. unitatea de decodificare a instrucțiunii (engl. *Instruction Decode Unit*);
4. unitatea de execuție (EU, engl. *Execution Unit*);

5. unitatea de translatare a adreselor (engl. *Segment Unit*);
6. unitatea de paginare (engl. *Paging Unit*).

Mai multe detalii despre paralelismul intern la nivelul procesorului se pot consulta în [13].

Rezumat.

Acest paragraf a încercat să atragă atenția studenților că *arhitectura calculatoarelor* este o poartă care deschide reale perspective de documentare şi însuşire de cunoştințe în domenii moderne ca: rețele de calculatoare, sisteme distribuite, prelucrare nesecvențială (paralelă şi/sau concurentă). Conceptele nu sunt tratate aici exhaustiv, fiecare dintre ele reprezentând în fond obiectul unei alte discipline de studiu.

Verificare.

1. Care sunt avantajele prelucrării în paralel a sarcinilor de calcul şi/sau de control?
2. Care sunt principalele unități funcționale ale calculatorului care lucrează în paralel?
3. Comparați structura benzii de asamblare a procesorului în cinci stagii (paragraful *Moduri de adresare*) cu funcțiile celor şase unități interne de prelucrare ale procesorului descrise în finalul paragrafului.

NOTIȚE